IBM WebSphere Application Server for Distributed Platforms, Version 8.5

Securing applications and their environment

Note

Before using this information, be sure to read the general information under “Notices” on page 1053.

Compilation date: June 4, 2012

Contents

How to send your comments

Your feedback is important in helping to provide the most accurate and highest quality information.

- To send comments on articles in the WebSphere Application Server Information Center
  1. Display the article in your Web browser and scroll to the end of the article.
  2. Click on the **Feedback** link at the bottom of the article, and a separate window containing an email form appears.
  3. Fill out the email form as instructed, and submit your feedback.
- To send comments on PDF books, you can email your comments to: **wasdoc@us.ibm.com**.

  Your comment should pertain to specific errors or omissions, accuracy, organization, subject matter, or completeness of this book. Be sure to include the document name and number, the WebSphere Application Server version you are using, and, if applicable, the specific page, table, or figure number on which you are commenting.

For technical questions and information about products and prices, please contact your IBM branch office, your IBM business partner, or your authorized remarketer. When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you. IBM or any other organizations will only use the personal information that you supply to contact you about your comments.

Using this PDF

Links

Because the content within this PDF is designed for an online information center deliverable, you might experience broken links. You can expect the following link behavior within this PDF:

- Links to Web addresses beginning with http:// work.
- Links that refer to specific page numbers within the same PDF book work.
- The remaining links will *not* work. You receive an error message when you click them.

Print sections directly from the information center navigation

PDF books are provided as a convenience format for easy printing, reading, and offline use. The information center is the official delivery format for IBM WebSphere Application Server documentation. If you use the PDF books primarily for convenient printing, it is now easier to print various parts of the information center as needed, quickly and directly from the information center navigation tree.

To print a section of the information center navigation:

1. Hover your cursor over an entry in the information center navigation until the **Open Quick Menu** icon is displayed beside the entry.
2. Right-click the icon to display a menu for printing or searching your selected section of the navigation tree.
3. If you select **Print this topic and subtopics** from the menu, the selected section is launched in a separate browser window as one HTML file. The HTML file includes each of the topics in the section, with a table of contents at the top.
4. Print the HTML file.

For performance reasons, the number of topics you can print at one time is limited. You are notified if your selection contains too many topics. If the current limit is too restrictive, use the feedback link to suggest a preferable limit. The feedback link is available at the end of most information center pages.

Chapter 1. Overview and new features for securing applications and their environment

Use the links provided in this topic to learn more about the security infrastructure.

What is new for security specialists

This topic provides an overview of new and changed features in security.

Security

This topic describes how IBM® WebSphere® Application Server provides security infrastructure and mechanisms to protect sensitive Java Platform, Enterprise Edition (Java EE) resources and administrative resources and to address enterprise end-to-end security requirements on authentication, resource access control, data integrity, confidentiality, privacy, and secure interoperability.

"Security planning overview"

Several communication links are provided from a browser on the Internet, through web servers and product servers, to the enterprise data at the back-end. This topic examines some typical configurations and common security practices. WebSphere Application Server security is built on a layered security architecture. This section also examines the security protection offered by each security layer and common security practice for good quality of protection in end-to-end security.

Samples

The Samples documentation offers:

- **Login - Form Login**

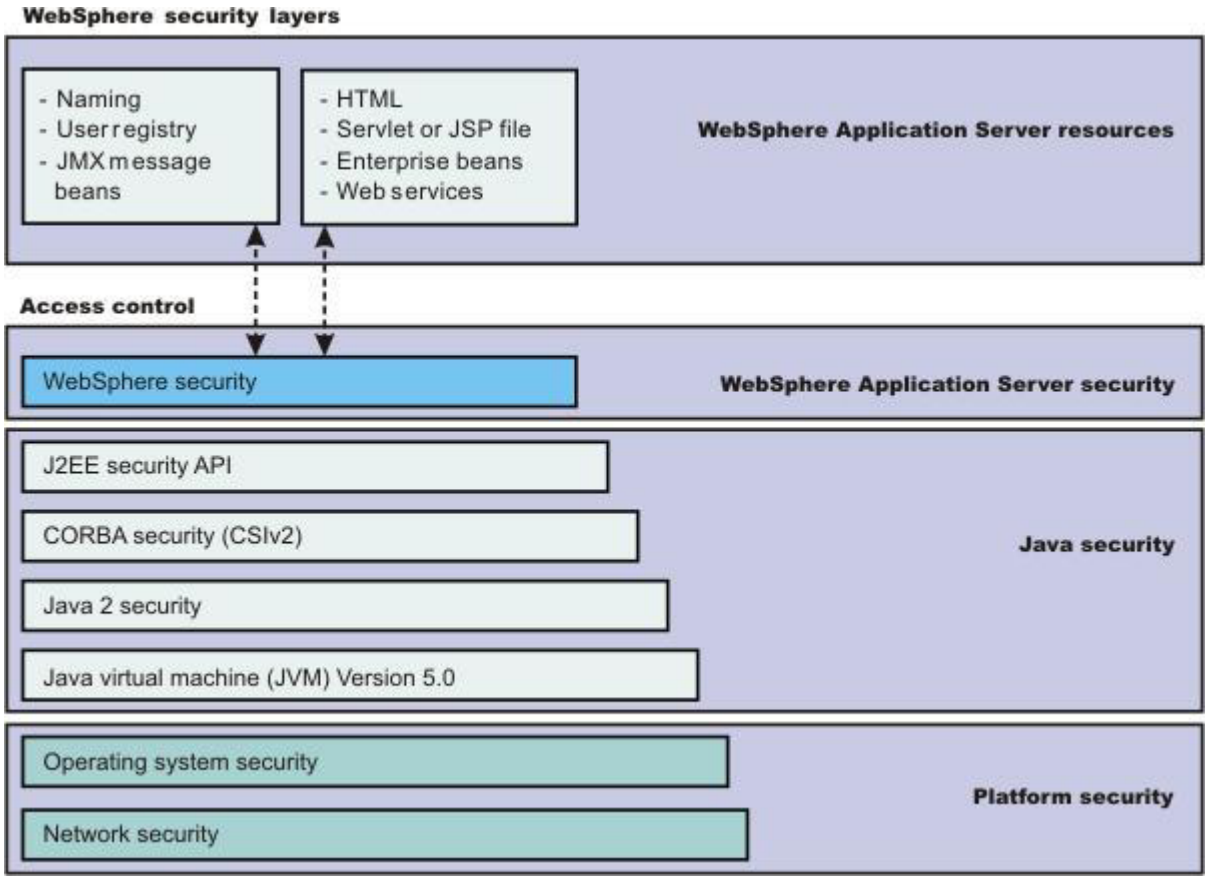
  The Form Login Sample demonstrates a very simple example of how to use the login facilities for WebSphere Application Server to implement and configure login applications. The Sample uses the Java Platform, Enterprise Edition (Java EE) form-based login technology to customize the look and feel of the login screens. It uses servlet filters to log the user information and the date information. The Sample finishes the session by using the form-based logout function, an IBM extension to the Java EE specification.
- **Login - JAAS Login**

  The JAAS Login Sample demonstrates how to use the Java Authentication and Authorization Service (JAAS) with WebSphere Application Server. The Sample uses server-side login with JAAS to authenticate a real user to the WebSphere security run time. Based upon a successful login, the WebSphere security run time uses the authenticated Subject to perform authorization checks on a protected stateless session enterprise bean. If the Sample runs successfully, it displays all the principals and public credentials of the authenticated user.

Security planning overview

When you access information on the Internet, you connect through web servers and product servers to the enterprise data at the back end. This section examines some typical configurations and common security practices.

This section also examines the security protection that is offered by each security layer and common security practice for good quality of protection in end-to-end security. The following figure illustrates the building blocks that comprise the operating environment for security within WebSphere Application Server:



The following information describes each of the components of WebSphere Application Server security, Java security, and Platform security that are illustrated in the previous figure.

WebSphere Application Server security

WebSphere security
WebSphere Application Server security enforces security policies and services in a unified manner on access to Web resources, enterprise beans, and JMX administrative resources. It consists of WebSphere Application Server security technologies and features to support the needs of a secure enterprise environment.

Java security

Java Platform, Enterprise Edition (Java EE) security application programming interface (API)
The security collaborator enforces Java Platform, Enterprise Edition (Java EE)-based security policies and supports Java EE security APIs.

Java 2 security
The Java 2 Security model offers fine-grained access control to system resources including file system, system property, socket connection, threading, class loading, and so on. Application code must explicitly grant the required permission to access a protected resource.

Java Virtual Machine (JVM) 5.0
The JVM security model provides a layer of security above the operating system layer. For example, JVM security protects the memory from unrestricted access, creates exceptions when errors occur within a thread, and defines array types.

Platform security

Operating system security

The security infrastructure of the underlying operating system provides certain security services for WebSphere Application Server. These services include the file system security support that secures sensitive files in the product installation for WebSphere Application Server. The system administrator can configure the product to obtain authentication information directly from the operating system user registry.

The security infrastructure of the underlying operating system provides certain security services for WebSphere Application Server. The operating system identity of the servant, controller, and daemon Started Task, as established by the STARTED profile, is the identity that is used to control access to system resources such as files or sockets. Optionally, the operating system security can provide authentication services using the User Registry of local operating system, and/or authorization services using SAF Authorization for the WebSphere Administration console and for applications running under the application server.

In addition to knowledge of Secure Sockets Layer (SSL) and Transport Layer Security (TLS), the administrator must be familiar with System Authorization Facility (SAF) and Resource Access Control Facility (RACF®), or an equivalent SAF based product.

The identity and verification of users can be managed by using a Local Operating System as the User Registry, RACF or equivalent SAF base product. Alternatively, an LDAP, Custom, or Federated User Registry can be used.

WebSphere can be configured to use SAF Authorization, which will use RACF or an equivalent SAF based product to manage and protect users and group resources. Alternatively, WebSphere can be configured to use WebSphere Authorization or a JACC External Authorization Provider.

When using either Local Operating System as the User Registry and/or using SAF Authorization, security auditing is an inherit feature of RACF or the equivalent SAF based products.

Network security

The Network Security layers provide transport level authentication and message integrity and confidentiality. You can configure the communication between separate application servers to use Secure Sockets Layer (SSL). Additionally, you can use IP Security and Virtual Private Network (VPN) for added message protection.

Each product application server consists of a web container, an Enterprise Java Beans (EJB) container, and the administrative subsystem.

The administrative console is a special Java EE web application that provides the interface for performing administrative functions. WebSphere Application Server configuration data is stored in XML descriptor files, which must be protected by operating system security. Passwords and other sensitive configuration data can be modified using the administrative console. However, you must protect these passwords and sensitive data. For more information, see “Encoding passwords in files” on page 983.

The administrative console web application has a setup data constraint that requires access to the administrative console servlets and JavaServer Pages (JSP) files only through an SSL connection when administrative security is enabled.

In WebSphere Application Server Version 6.0.x and earlier, the administrator console HTTPS port was configured to use `DummyServerKeyFile.jks` and `DummyServerTrustFile.jks` with the default self- signed certificate. The dummy certificates and keys must be replaced immediately after WebSphere Application Server installation; the keys are common in all of the installation and are therefore insecure. WebSphere Application Server Version 6.1 provides integrated certificate and key management, which generate distinct private key and self-signed certificate with embedded server host name to enable host name verification.

WebSphere Application Server Version 6.1 also enables integration with external certificate (CA) authority to use CA-issued certificates. The WebSphere Application Servers Version 6.1 installation process provides an option to enable administrative security during installation. As a result, a WebSphere Application Server process is secured immediately after installation. WebSphere Application Server Version 7.0 extends the embedded certificate management capabilities by creating a chained certificate (personal certificate signed by a root certificate) to enable refresh of the personal certificate without affecting the trust established. It also enables tailoring of the certificate during profile creation (you can import your own or change the distinguished name (DN) of the one created by default) as well as the ability to change the default keystore password.

Administrative security

WebSphere Application Servers interact with each other through CSIv2 and Secure Authentication Services (SAS) security protocols as well as the HTTP and HTTPS protocols.

Important: SAS is supported only between Version 6.0.x and previous version servers that have been federated in a Version 6.1 cell.

You can configure these protocols to use Secure Sockets Layer (SSL) when you enable WebSphere Application Server administrative security. The WebSphere Application Server administrative subsystem in every server uses SOAP, Java Management Extensions (JMX) connectors and Remote Method Invocation over the Internet Inter-ORB Protocol (RMI/IIOP) JMX connectors to pass administrative commands and configuration data. When administrative security is disabled, the SOAP JMX connector uses the HTTP protocol and the RMI/IIOP connector uses the TCP/IP protocol. When administrative security is enabled, the SOAP JMX connector always uses the HTTPS protocol. When administrative security is enabled, you can configure the RMI/IIOP JMX connector to either use SSL or to use TCP/IP. It is recommended that you enable administrative security and enable SSL to protect the sensitive configuration data.

Security for Java EE resources

Security for Java EE resources is provided by the web container and the EJB container. Each container provides two kinds of security: declarative security and programmatic security.

In declarative security, an application security structure includes network message integrity and confidentiality, authentication requirements, security roles, and access control. Access control is expressed in a form that is external to the application. In particular, the deployment descriptor is the primary vehicle for declarative security in the Java EE platform. WebSphere Application Server maintains Java EE security policy, including information that is derived from the deployment descriptor and specified by deployers and administrators in a set of XML descriptor files. At runtime, the container uses the security policy that is defined in the XML descriptor files to enforce data constraints and access control.

When declarative security alone is not sufficient to express the security model of an application, you might use programmatic security to make access decisions. When administrative security is enabled and application server security is not disabled at the server level, Java EE applications security is enforced. When the security policy is specified for a web resource, the web container performs access control when the resource is requested by a web client. The web container challenges the web client for authentication data if none is present according to the specified authentication method, ensures that the data constraints are met, and determines whether the authenticated user has the required security role. The web security collaborator enforces role-based access control by using an access manager implementation. An access manager makes authorization decisions that are based on security policy derived from the deployment descriptor. An authenticated user principal can access the requested servlet or JSP file if the user principal has one of the required security roles. Servlets and JSP files can use the HttpServletRequest methods, isUserInRole and getUserPrincipal.

When administrative security and application security are enabled, and the application server level application security is not disabled, the EJB container enforces access control on EJB method invocation.

The authentication occurs regardless of whether method permission is defined for the specific EJB method. The EJB security collaborator enforces role-based access control by using an access manager implementation. An access manager makes authorization decisions that are based on security policy derived from the deployment descriptor. An authenticated user principal can access the requested EJB method if it has one of the required security roles. EJB code can use the EJBContext methods, isCallerInRole and getCallerPrincipal. Use the Java EE role-based access control to protect valuable business data from access by unauthorized users through the Internet and the intranet. Refer to Securing web applications using an assembly tool, and Securing enterprise bean applications.

Role-based security

WebSphere Application Server extends the security, role-based access control to administrative resources including the JMX system management subsystem, user registries, and Java Naming and Directory Interface (JNDI) name space. WebSphere administrative subsystem defines four administrative security roles:

Monitor role
A monitor can view the configuration information and status but cannot make any changes.

Operator role
An operator can trigger run-time state changes, such as start an application server or stop an application but cannot make configuration changes.

Configurator role
A configurator can modify the configuration information but cannot change the state of the runtime.

Administrator role
An operator as well as a configurator, which additionally can modify sensitive security configuration and security policy such as setting server IDs and passwords, enable or disable administrative security and Java 2 security, and map users and groups to the administrator role.

iscadmins
The iscadmins role has administrator privileges for managing users and groups from within the administrative console only.

WebSphere Application Server defines two additional roles that are available when you use wsadmin scripting only.

Deployer
A deployer can perform both configuration actions and run-time operations on applications.

Adminsecuritymanager
An administrative security manager can map users to administrative roles. Also, when fine grained admin security is used, users granted this role can manage authorization groups.

Auditor
An auditor can view and modify the configuration settings for the security auditing subsystem.

A user with the configurator role can perform most administrative work including installing new applications and application servers. Certain configuration tasks exist that a configurator does not have sufficient authority to do when administrative security is enabled, including modifying a WebSphere Application Server identity and password, Lightweight Third-Party Authentication (LTPA) password and keys, and assigning users to administrative security roles. Those sensitive configuration tasks require the administrative role because the server ID is mapped to the administrator role.

Enable WebSphere Application Server administrative security to protect administrative subsystem integrity. Application server security can be selectively disabled if no sensitive information is available to protect. For securing administrative security, refer to “Authorizing access to administrative roles” on page 632 and Assigning users and groups to roles.

Java 2 security permissions

WebSphere Application Server uses the Java 2 security model to create a secure environment to run application code. Java 2 security provides a fine-grained and policy-based access control to protect system resources such as files, system properties, opening socket connections, loading libraries, and so on. The Java EE Version 1.4 specification defines a typical set of Java 2 security permissions that web and EJB components expect to have.

Table 1. Java EE security permissions set for web components. The Java EE security permissions set for web components are shown in the following table.

Security Permission	Target	Action
java.lang.RuntimePermission	loadLibrary	
java.lang.RuntimePermission	queuePrintJob	
java.net.SocketPermission	*	connect
java.io.FilePermission	*	read, write
java.util.PropertyPermission	*	read

Table 2. Java EE security permissions set for EJB components. The Java EE security permissions set for EJB components are shown in the following table.

Security Permission	Target	Action
java.lang.RuntimePermission	queuePrintJob	
java.net.SocketPermission	*	connect
java.util.PropertyPermission	*	read

The WebSphere Application Server Java 2 security default policies are based on the Java EE Version 1.4 specification. The specification grants web components read and write file access permission to any file in the file system, which might be too broad. The WebSphere Application Server default policy gives web components read and write permission to the subdirectory and the subtree where the web module is installed. The default Java 2 security policies for all Java virtual machines and WebSphere Application Server processes are contained in the following policy files:

${java.home}/jre/lib/security/java.policy
This file is used as the default policy for the Java virtual machine (JVM).

${USER_INSTALL_ROOT}/properties/server.policy
This file is used as the default policy for all product server processes.

To simplify policy management, WebSphere Application Server policy is based on resource type rather than code base (location). The following files are the default policy files for a WebSphere Application Server subsystem. These policy files, which are an extension of the WebSphere Application Server runtime, are referred to as Service Provider Programming Interfaces (SPI), and shared by multiple Java EE applications:

- *profile_root*/config/cells/*cell_name*/nodes/*node_name*/spi.policy
  This file is used for embedded resources defined in the resources.xml file, such as the Java Message Service (JMS), JavaMail, and JDBC drivers.
- *profile_root*/config/cells/*cell_name*/nodes/*node_name*/library.policy
  This file is used by the shared library that is defined by the WebSphere Application Server administrative console.
- *profile_root*/config/cells/*cell_name*/nodes/*node_name*/app.policy
  This file is used as the default policy for Java EE applications.

In general, applications do not require more permissions to run than those recommended by the Java EE specification to be portable among various application servers. However, some applications might require

more permissions. WebSphere Application Server supports the packaging of a `was.policy` file with each application to grant extra permissions to that application.

Attention: Grant extra permissions to an application only after careful consideration because of the potential of compromising the system integrity.

Loading libraries into WebSphere Application Server does allow applications to leave the Java sandbox. WebSphere Application Server uses a permission filtering policy file to alert you when an application installation fails because of additional permission requirements. For example, it is recommended that you not give the java.lang.RuntimePermission exitVM permission to an application so that application code cannot terminate WebSphere Application Server.

The filtering policy is defined by the filtermask in the *profile_root*`/config/cells/`*cell_name*`/filter.policy` file. Moreover, WebSphere Application Server also performs run-time permission filtering that is based on the run-time filtering policy to ensure that application code is not granted a permission that is considered harmful to system integrity.

Therefore, many applications developed for prior releases of WebSphere Application Server might not be Java 2 security ready. To quickly migrate those applications to the latest version of WebSphere Application Server, you might temporarily give those applications the java.security.AllPermission permission in the `was.policy` file. Test those applications to ensure that they run in an environment where Java 2 security is active. For example, identify which extra permissions, if any, are required, and grant only those permissions to a particular application. Not granting the `AllPermission` permission to applications can reduce the risk of compromising system integrity. For more information on migrating applications, refer to "Migrating Java 2 security policy" on page 62.

The WebSphere Application Server runtime uses Java 2 security to protect sensitive run-time functions. Applications that are granted the `AllPermission` permission not only have access to sensitive system resources, but also WebSphere Application Server run-time resources and can potentially cause damage to both. In cases where an application can be trusted as safe, WebSphere Application Server does support having Java 2 security disabled on a per application server basis. You can enforce Java 2 security by default in the administrative console and clear the Java 2 security flag to disable it at the particular application server.

When you specify the `Enable administrative security` and Use Java 2 security to restrict application access to local resources options on the Global security panel of the administrative console, the information and other sensitive configuration data, are stored in a set of XML configuration files. Both role-based access control and Java 2 security permission-based access control are employed to protect the integrity of the configuration data. The example uses configuration data protection to illustrate how system integrity is maintained.

Attention: The `Enable global security` option in previous releases of WebSphere Application Server is the same as the `Enable administrative security` option in Version 8.5. Also, the Enable Java 2 security option in previous releases is the same as the Use Java 2 security to restrict application access to local resources option in Version 8.5.

- When Java 2 security is enforced, the application code cannot access the WebSphere Application Server run-time classes that manage the configuration data unless the code is granted the required WebSphere Application Server run-time permissions.
- When Java 2 security is enforced, application code cannot access the WebSphere Application Server configuration XML files unless the code is granted the required file read and write permission.
- The JMX administrative subsystem provides SOAP over HTTP or HTTPS and a RMI/IIOP remote interface to enable application programs to extract and to modify configuration files and data. When administrative security is enabled, an application program can modify the WebSphere Application Server configuration if the application program has presented valid authentication data and the security identity has the required security roles.

- If a user can disable Java 2 security, the user can also modify the WebSphere Application Server configuration, including the WebSphere Application Server security identity and authentication data with other sensitive data. Only users with the administrator security role can disable Java 2 security.
- Because WebSphere Application Server security identity is given to the administrator role, only users with the administrator role can disable administrative security, change server IDs and passwords, and map users and groups to administrative roles, and so on.

Other Runtime resources

Other WebSphere Application Server run-time resources are protected by a similar mechanism, as described previously. It is very important to enable WebSphere Application Server administrative security and to use Java 2 security to restrict application access to local resources. Java EE Specification defines several authentication methods for web components: HTTP Basic Authentication, Form-Based Authentication, and HTTPS Client Certificate Authentication. When you use client certificate login, it is more convenient for the browser client if the web resources have integral or confidential data constraint. If a browser uses HTTP to access the web resource, the web container automatically redirects the browser to the HTTPS port. The CSIv2 security protocol also supports client certificate authentication. You can also use SSL client authentication to set up secure communication among a selected set of servers based on a trust relationship.

If you start from the WebSphere Application Server plug-in at the web server, you can configure SSL mutual authentication between it and the WebSphere Application Server HTTPS server. When using a certificate, you can restrict the WebSphere Application Server plug-in to communicate with only the selected two WebSphere Application Servers as shown in the following figure. Note that you can use self-signed certificates to reduce administration and cost.

For example, you want to restrict the HTTPS server in WebSphere Application Server A and in WebSphere Application Server B to accept secure socket connections only from the WebSphere Application Server plug-in W.

- To complete this task, you can generate three certificates using the IKEYMAN and the certificate management utilities. Also, you can use certificate W and trust certificate A and B. Configure the HTTPS server of WebSphere Application Server A to use certificate A and to trust certificate W.

Configure the HTTPS server of WebSphere Application Server B to use certificate B and to trust certificate W.

Table 3. Trust relationships from example. The trust relationship that is depicted in the previous figure is shown in the following table.

Server	Key	Trust
WebSphere Application Server plug-in	W	A, B
WebSphere Application Server A	A	W
WebSphere Application Server B	B	W

When WebSphere Application Server is configured to use Lightweight Directory Access Protocol (LDAP) user registry, you also can configure SSL with mutual authentication between every application server and the LDAP server with self-signed certificates so that a password is not visible when it is passed from WebSphere Application Server to the LDAP server.

WebSphere Application Server does not provide a registry configuration or management utility. In addition, it does not dictate the registry password policy. It is recommended that you use the password policy recommended by your registry, including the password length and expiration period.

Before securing your WebSphere Application Server environment, determine which versions of WebSphere Application Server you are using, review the WebSphere Application Server security architecture, and review each of the following topics:

- “Common Secure Interoperability Version 2 features” on page 522
- “Identity assertion to the downstream server” on page 523
- “Selecting an authentication mechanism” on page 341
  - “Lightweight Third Party Authentication” on page 343
  - “Trust associations” on page 364
  - “Single sign-on for authentication using LTPA cookies” on page 370
- “Selecting a registry or repository” on page 159
  - “Local operating system registries” on page 164
  - “Standalone Lightweight Directory Access Protocol registries” on page 337
- “Java 2 security” on page 74
  - “Java 2 security policy files” on page 79
- “Java Authentication and Authorization Service” on page 437
  - Programmatic login for JAAS
- Java EE connector security
- “Access control exception for Java 2 security” on page 83
  - “Role-based authorization” on page 571
  - “Administrative roles and naming service authorization” on page 566
- “Implementing a custom authentication provider using JASPI” on page 928

Chapter 2. Securing the Liberty profile and its applications

This information applies generally to all types of applications deployed on the Liberty profile.

About this task

Security in the Liberty profile supports all the Servlet 3.0 security features. In addition, it also secures Java JMX connections. The following server features are applicable to security in the Liberty profile:

- `appSecurity-1.0` enables security for all web resources.
- `ssl-1.0` enables SSL connections using HTTPS.
- `restConnector-1.0` enables remote access by JMX client through a REST-based connector.

To learn about how security works in the Liberty profile, see Liberty profile: Security.

There are several security configuration examples under the `/templates/config` directory of the server image for reference when configuring security for your applications on the Liberty profile.

Best practice: When you use the developer tools to configure the security on the Liberty profile, make sure that the configuration created by the tools is similar to the examples in the `${wlp.install.dir}/templates/config` directory of the server image. This directory includes examples of configuring some of the most common security features. If you see any differences in the configuration created by the developer tools and the examples, modify the configuration to fit the configuration in the examples for that feature.

Procedure

- Use `quickStartSecurity` for minimal security configuration
- Secure communication with the Liberty profile
- Access secured JMX connector on the Liberty profile
- Authenticate users in the Liberty profile
- Authorize access to resources in the Liberty profile
- Secure a database access application
- Develop extensions to the Liberty profile security infrastructure

Getting started with security in the Liberty profile

You can use the `quickStartSecurity` element to quickly enable a simple (one user) security setup for the Liberty profile.

About this task

This topic goes through the basic steps required to set up a secured Liberty profile server and web application. Additionally, configuration actions within the Liberty profile are dynamic, which means the configuration updates take effect without having to restart the server.

Procedure

1. Create and start your server.
   - Windows On Windows systems:

     ```
     bin\server.bat create MyNewServer
     bin\server.bat start MyNewServer
     ```

   - AIX Linux AIX HP-UX Solaris HP-UX Solaris On all systems other than Windows systems:

```
bin/server create MyNewServer
bin/server start MyNewServer
```

2. Include the `appSecurity-1.0` feature in the `server.xml` file. The `server.xml` file is located in the server directory of *myNewServer*, for example, `wlp\usr\servers\myNewServer\server.xml`.

```
<featureManager>
     <feature>appSecurity-1.0</feature>
</featureManager>
```

3. Define the user name and password that is to be granted the **Administrator** role for server management activities.

```
<quickStartSecurity userName="Bob" userPassword="bobpwd" />
```

Note: Choose a user name and password that are meaningful to you. Never use the name and password in the example for your applications.

4. Configure the deployment descriptor with the relevant security constraints to protect the web resource. For example, use `<auth-constraint>` and `<role-name>` elements to define a role that is allowed to access the web resource.

   The following example `web.xml` file shows that access to all the URIs in the application is protected by the `testing` role.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
                         "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app id="myWebApp">

 <!-- SERVLET DEFINITIONS -->
 <servlet id="Default">
    <servlet-name>myWebApp</servlet-name>
    <servlet-class>com.web.app.MyWebAppServlet</servlet-class>
    <load-on-startup/>
 </servlet>

 <!-- SERVLET MAPPINGS -->
 <servlet-mapping id="ServletMapping_Default">
    <servlet-name>myWebApp</servlet-name>
    <url-pattern>/*</url-pattern>
 </servlet-mapping>

 <!-- SECURITY ROLES -->
 <security-role>
    <role-name>testing</role-name>
 </security-role>

 <!-- SECURITY CONSTRAINTS -->
 <security-constraint>
    <web-resource-collection>
      <url-pattern>/*</url-pattern>
    </web-resource-collection>
    <auth-constraint>
      <role-name>testing</role-name>
    </auth-constraint>
 </security-constraint>

 <!-- AUTHENTICATION METHOD: Basic authentication -->
 <login-config>
    <auth-method>BASIC</auth-method>
 </login-config>

</web-app>
```

5. Configure your application in the `server.xml` file.

   In the following example, the user `Bob` is mapped to the `testing` role of the application:

```
<application type="war" id="myWebApp" name="myWebApp"
             location="${server.config.dir}/apps/myWebApp.war">
    <application-bnd>
        <security-role name="testing">
             <user name="Bob" />
        </security-role>
    </application-bnd>
</application>
```

6. Access your application and log in with the user name `Bob`. The default URL for the `myWebApp` application is `http://localhost:9080/myWebApp`

Results

You have now secured your application.

Liberty profile: Quick overview of security

This topic describes some of common security terms, along with an example which helps you understand the basic workflow of security in the Liberty profile.

Security key terms

Authorization
: The process of determining whether or not to grant a user access to resources within the system is known as authorization. The Java EE model uses subjects, resources and roles to determine what should and should not be allowed.

Authentication
: The process of confirming the identity of a user is known as authentication. The most common form of authentication is user name and password, such as through either basic authentication or form login for web applications. Once a user is authenticated, the source of a request is represented as a `Subject` object at the run time.

Resource
: Also known as an object, resources are things within the system. A resource can be any non-active entity, such as a web application.

Role
: A role is a logical collection of privileges that can be assigned to a user or group. Some roles are predefined by the system (such as the Administrator role). Others are defined by the application developer. In Java EE, subjects are usually granted or denied access to resources based on the roles they do (or do not) possess.

Subject
: A subject is both a general term, as well as a Java object `javax.security.auth.Subject`. Generally, the term subject means active entities within the system, such as users on the system, and even the system process itself.

Security workflow

The following example demonstrates how the security works when a user requests access to a resource. For example, a user **Bob** wants to access a servlet `myWebApp`. See the code samples in “Getting started with security in the Liberty profile” on page 11.

In order to do this, the following conditions must be true:

1. **Bob** must be able to log into the system because the servlet is protected.
2. **Bob** must be in the `testing` role because the servlet is restricted using an `auth-constraint` element in the deployment descriptor.

If **Bob** cannot log into the system, or **Bob** is not in the `testing` role, then the access to the servlet `myWebApp` is denied.

Another user **Alice** can log into the system because **Alice** is a valid user. But **Alice** is not in the `testing` role. An HTTP 403 error (Access Denied/Forbidden) shows up when **Alice** logs in.

Setting up BasicRegistry and role mapping on the Liberty profile

You can configure the Liberty profile to authenticate and authorize users using a basic user registry.

Before you begin

The server feature appSecurity-1.0 must be enabled in the `server.xml` file of the Liberty profile.

About this task

This topic goes through the steps to set up a basic user registry and configure more role mapping in the `server.xml` file for a Liberty profile server.

Procedure

1. Configure the basic registry as follows. Make sure to use a user name and password that are meaningful to you. Never use the name and password in the example for your applications.

```
<basicRegistry id="basic" realm="WebRealm">
  <user name="Bob" password="bobpwd" />
</basicRegistry>
```

2. Optional: Grant the user with the **Administrator** role if the user is used to perform remote system management activities. This step is done automatically when using `quickStartSecurity` element.

```
<administrator-role>
   <user>Bob</user>
</administrator-role>
```

3. Encode the password within the configuration. You can get the encoded value by using the securityUtility encode task.
4. Add additional users. Make sure each user name is unique.

```
<basicRegistry id="basic" realm="WebRealm">
   <user name="Bob" password="bobpwd" />
  <user name="user1" password="user1pwd" />
  <user name="user2" password="user2pwd" />
</basicRegistry>
```

5. Create groups for users. Make sure each group name must be unique.

```
<basicRegistry id="basic" realm="WebRealm">
   <user name="Bob" password="bobpwd" />
  <user name="user1" password="user1pwd" />
  <user name="user2" password="user2pwd" />

       <group name="myAdmins">
          <member name="Bob" />
          <member name="user1" />
       </group>

       <group name="users">
          <member name="user1" />
          <member name="user2" />
       </group>
</basicRegistry>
```

6. Assign a user group to the **Administrator** role.

```
<administrator-role>
   <user>Bob</user>
   <group>myAdmins</group>
</administrator-role>
```

7. Assign some users and groups to the **testing** role of an application.

```
<application type="war" id="myWebApp" name="myWebApp"
             location="${server.config.dir}/apps/myWebApp.war">
   <application-bnd>
       <security-role name="tesing">
           <user name="Bob" />
           <user name="user1" />
           <group name="users" />
       </security-role>
   </application-bnd>
</application>
```

What to do next

Configure security related elements in the deployment descriptor of your application. See “Getting started with security in the Liberty profile” on page 11 for a sample `web.xml` file.

Securing communications with the Liberty profile

You can configure the Liberty profile server to provide secure communications between a client and the server.

About this task

To configure secure communications, you can either specify a minimal SSL configuration or a detailed SSL configuration in the `server.xml` file. The minimal configuration only requires the SSL feature and a keystore entry to be specified. In the `samples` directory of the Liberty profile, there is an `sslConfig.xml` file that contains several examples of SSL configurations.

The following topics are covered in this section:

Procedure

- Enable SSL communications between a client and a Liberty profile server
- Optional: Create a keystore from the command prompt
- Optional: Encode passwords from the command prompt
- Optional: Configure client certificate authentication between your application and the Liberty profile server

Enabling SSL communication for the Liberty profile

To enable SSL communication for the Liberty profile, there is a minimal set of SSL configuration options. It assumes most of the SSL options and only requires some keystore configuration information.

About this task

SSL client authentication occurs during the connection handshake using SSL certificates. The SSL handshake is a series of messages that are exchanged over the SSL protocol to negotiate for connection-specific protection. During the handshake, the secure server requests that the client send back a certificate or certificate chain for the authentication. To do this, you add the `ssl-1.0` server feature to the `server.xml` file, along with code that tells the server the keystore information for authentication.

Procedure

1. Enable the appSecurity-1.0 and ssl-1.0 server features in the server.xml file.

```
<featureManager>
    <feature>appSecurity-1.0</feature>
    <feature>ssl-1.0</feature>
</featureManager>
```

2. Add the keystore service object entry to the server.xml file. The keyStore element is called defaultKeyStore and contains the keystore password. The password can be entered in clear text or encoded. The securityUtility encode option can be used to encode the password.

```
<keyStore id="defaultKeyStore" password="yourPassword" />
```

   Avoid trouble: When using the developer tools to create a minimal SSL configuration, make sure to enter defaultKeyStore in the **id** field and a password. Otherwise, the SSL configuration fails and the services using this configuration fails to start. For example, if the httpEndpoint element is using this SSL configuration, the HTTPS port doesn't start.

   In this configuration the keystore type is JKS. You can create this default keystore using the securityUtility createSSLCertificate option, the server creates the keystore for you if it does not exist during SSL initialization. The password must be at least 6 characters long. The type of the keystore is JKS by default. Keystore of other types can also be specified in the minimal SSL configuration if the keystore file is already created. Only JKS keystore files are created by the server if the keystore file does not exist. The certificate has a validity period of 365 days, the CN value of the subjectDN is the hostname of the machine where the server is running, and the signature algorithm of the certificate is SHA1 with RSA.

   The single keystore entry for a minimal SSL configuration can be extended to include the location and type as well.

```
<keyStore id="defaultKeyStore" location="myKeyStore.p12" password="yourPassword" type="PKCS12"/>
```

   The **location** parameter can be an absolute path to the keystore file. If it is an absolute path, then the keystore file is assumed to have been already created. Keystore of other types can also be specified in the minimal SSL configuration as long as the keystore file is already created. When the minimal SSL configuration is used, the SSL configuration defaults are used to create the SSL context for an SSL handshake. The configuration protocol is SSL_TLS by default. The HIGH ciphers, 128 bit and higher cipher suites can be used.

Liberty profile: SSL configuration attributes

SSL configurations contain attributes that you use to control the behavior of the server SSL transport layer on a Liberty profile. This document iterates all the settings available for an SSL configuration.

SSL Feature

To enable SSL on a server, the SSL feature must be included in the server.xml file:

```
<featureManager>
  <feature>ssl-1.0</feature>
</featureManager>
```

SSL Default

You can have multiple SSL configurations configured. If more than one is configured, then the default SSL configuration must be specified in the server.xml file using the sslDefault service configuration.

Table 4. Attribute of the SSLDefault element. This table describes the attribute of the `SSLDefault` element.

Attribute	Description	Default Value
sslRef	The **sslRef** attribute specifies the SSL configuration to be used as the default. If this attribute is not specified, then the value used is `defaultSSLSettings`.	The default SSL Configuration name is `defaultSSLSettings`.

In the `server.xml` file, the entry looks like this:

```
<sslDefault sslRef="mySSLSettings" />
```

SSL Configuration

You use the SSL configuration attributes to customize the SSL environment to suit your needs. These attributes can be set on the `ssl` service configuration element in the `server.xml` file.

Table 5. Attributes of the SSL element. This table describes the attributes of the `ssl` element.

Attribute	Description	Default Value
id	The **id** attribute assigns a unique name to the SSL configuration object.	No default value; a unique name must be specified.
keyStoreRef	The **keyStoreRef** attribute names the keystore service object that defines the SSL configurations keystore. The keystore holds the key needed to make an SSL connection.	No default value; a keystore reference must be specified.
trustStoreRef	The **trustStoreRef** attribute names the keystore service object that defines the SSL configurations truststore. The truststore holds certificates needed for signing verification.	`trustStoreRef` is an optional attribute if the reference is missing. The keystore specified by `keyStoreRef` is used.
clientAuthentication	The **clientAuthentication** attribute determines whether SSL client authentication is required.	Default value is **false**.
clientAuthenticationSupported	The **clientAuthenticationSupported** attribute determines whether SSL client authentication is supported. The client does not have to supply a client certificate. If the **clientAuthentication** attribute is set to true, the value of the **clientAuthenticationSupported** attribute is overwritten.	Default value is **false**.
sslProtocol	The **sslProtocol** attribute defines the SSL handshake protocol. The protocol can be SDK dependent, so if modifying the protocol make sure the value is supported by the SDK you are running under.	Default value is **SSL_TLS**.

Table 5. Attributes of the SSL element (continued). This table describes the attributes of the `ssl` element.

Attribute	Description	Default Value
securityLevel	The **securityLevel** attribute determines the cipher suite group to be used by the SSL handshake. The attribute has one of the following values: • **HIGH** (128-bit ciphers and higher) • **MEDIUM** (40-bit ciphers) • **WEAK** (for all ciphers without encryption) • **CUSTOM** (if the cipher suite group is customized). When you set the **enableCiphers** attribute with a specific list of ciphers, the system ignores this attribute.	Default value is **HIGH**.
enableCiphers	The **enableCiphers** attribute is used to specify a unique list of cipher suites. Separate each cipher suite in the list with a space. If the **enableCiphers** attribute is set then the **securityLevel** attribute is ignored	No default value.
serverKeyAlias	The **serverKeyAlias** attribute names the key in the keystore to be used as the SSL configurations key. This attribute is only needed if the keystore has more than one key entry in it. If the keystore has more than one key entry and this attribute does not specify a key, then the JSSE picks a key.	No default value.
clientKeyAlias	The **clientKeyAlias**attribute names the key in the keystore to be used as the key for SSL configuration when **clientAuthentication** is enabled. The attribute is only required if the keystore contains more than one key entry.	No default value.

Note:

- The key manager is used by the SSL Handshake to determine what certificate alias to use. The key manager is not configured in the `server.xml` file, it is retrieved from the security property `ssl.KeyManagerFactory.algorithm` of the SDK.
- The trust manager is used by the SSL handshake to make trust decisions. The trust manager is not configured in the `server.xml` file, it is retrieved from the security property `ssl.TrustManagerFactory.algorithm` of the SDK.

Here is an example of how the `ssl` element is configured in the`server.xml` file:

```
<!--  Simple ssl configuration service object. This assumes there is a keystore object named -->
<!--  defaultKeyStore and a truststore object named defaultTrustStore in the server.xml file. -->
  <ssl id="myDefaultSSLConfig"
       keyStoreRef="defaultKeyStore"
       trustStoreRef="defaultTrustStore" />
```

```
<!--  A ssl configuration service object that enabled clientAuthentication -->
<!--  and specifies the TLS protocol be used. -->
  <ssl id="myDefaultSSLConfig"
       keyStoreRef="defaultKeyStore"
       trustStoreRef="defaultTrustStore"
       clientAuthentication="true"
       sslProtocol="TLS" />

<!--  A ssl configuration service object that names the serverKeyAlias to be use by the handshake. -->
<!--  This assumes there is a certificate called "default" in the keystore defined by keyStoreRef. -->
  <ssl id="myDefaultSSLConfig"
       keyStoreRef="defaultKeyStore"
       serverKeyAlias="default" />
```

Keystore Configuration

The `keystore` configuration consists of the attributes needed to load a keystore. These attribute can be set on the keystore service configuration in the `server.xml` file.

Table 6. Attributes of the keystore element. This table explains the attributes of `keystore` element.

Attribute	Description	Default Value
id	The **id** attribute defines a unique identifier of the keystore object.	No default value, a unique name must be specified.
location	The **location** attribute specifies the keystore file name. The value can include the absolute path to the file. If the absolute path is not provided, then the code looks for the file in the `${server.config.dir}/resources/security` directory.	In the SSL minimal configuration, the location of the file is assumed to be `${server.config.dir}/resources/security/key.jks`.
type	The **type** attribute specifies the type of the keystore. Check that the keystore type that you specify is supported by the SDK you are running on.	Default value is **jks**.
password	The **password** attribute specifies the password used to load the keystore file. The password can be stored either in clear text or encoded. For information about how to encode the password, see the securityUtility encode option.	Must be provided.
provider	The **provider** attributes specifies the provider to be used to load the keystore. Some keystore types required a provider other then the SDK default.	By default no provider is specified.
fileBased	The **fileBased** attribute specifies whether or not the keystore is file-based.	Default value is **true**.

Here is an example of how the `keystore` element is configured in the `server.xml` file:

```
<!-- A keystore object called defaultKeyStore provides a location, -->
<!-- type, and password. The MyKeyStoreFile.jks file is assumed -->
<!-- to be located in ${server.config.dir}/resources/security -->
  keyStore id="defaultKeyStore"
           location="MyKeyStoreFile.jks"
           type="JKS" password="myPassword" />
```

Full SSL Configuration Example

Here is an example of a full SSL configuration in the `server.xml` file. This example has the following SSL configurations:

- `defaultSSLSettings`
- `mySSLSettings`

By default, the SSL configuration is set to `defaultSSLSettings`.

```
<featureManager>
  <feature>ssl-1.0</feature>
</featureManager>


<!-- default SSL configuration is defaultSSLSettings ->
  <sslDefault sslRef="defaultSSLSettings" />
  <ssl id="defaultSSLSettings"
       keyStoreRef="defaultKeyStore"
       trustStoreRef="defaultTrustStore"
       clientAuthenticationSupported="true" />
  <keyStore id="defaultKeyStore"
            location="key.jks"
            type="JKS" password="defaultPWD" />
  <keyStore id="defaultTrustStore"
            location="trust.jks"
            type="JKS" password="defaultPWD" />

  <ssl id="mySSLSettings"
       keyStoreRef="myKeyStore"
       trustStoreRef="myTrustStore"
       clientAuthentication="true" />
  <keyStore id="LDAPKeyStore"
            location="${server.config.dir}/myKey.p12"
            type="PKCS12"
            password="{xor}CDo9Hgw=" />
  <keyStore id="LDAPTrustStore"
            location="${server.config.dir}/myTrust.p12"
            type="PKCS12"
            password="{xor}CDo9Hgw=" />
```

Creating SSL certificates for your Liberty profile using the Utilities menu

Using the Liberty profile Utilities menu in the developer tools, you can create an SSL certificate.

Procedure

1. In the Servers view, right-click your Liberty server profile, and select **Utilities** > **Create SSL Certificate...**.
2. On the Create SSL Certificate page, you can create a default secure socket layer (SSL) certificate to use with your server.
   a. In the **Keystore password** field, type a password for your SSL certificate.
   b. Click the **Specify validity period (days)** field, and specify the number of days you want the certificate to be valid for. Minimum length of time is 365 days.
   c. Click the **Specify subject (DN):** field, and provide a value for your SSL subject.
3. Click **Finish**.

Creating SSL certificates from the command prompt

You can use the **securityUtility** command to create a default SSL certificate for use by the Liberty profile configuration.

Procedure

1. Open a command prompt, then change directory to the `wlp` directory.
2. Create an SSL certificate.

   Run the following command. If you do not specify a server name or a password, the command does not run. See "Liberty profile: securityUtility command."

   ```
   bin/securityUtility createSSLCertificate --server server_name --password your_password
   ```

Results

You have created a default keystore `key.jks` for the specified server. The keystore file is located under the `/resources/security` directory of the specified server. If a default keystore already exists, the command does not execute successfully.

What to do next

You can configure your server to use the keystore and enable the SSL in the server configuration by adding the following lines to the server configuration file:

```
<featureManager>
    <feature>ssl-1.0</feature>
</featureManager>

<keyStore id="defaultKeyStore" password="keystore_password" />
```

See "Enabling SSL communication for the Liberty profile" on page 15.

Liberty profile: securityUtility command

The **securityUtility** command supports plain text encryption and SSL certificate creation for a Liberty profile.

Syntax

The command syntax is as follows:

```
securityUtility task [options]
```

where the **options** are different based on the value of **task**.

Parameters

The following tasks are available for the `securityUtility` command:

encode

Encodes the provided text using Base64 encryption. If no arguments are specified , the command enters interactive mode. Otherwise, the provided text is encoded. Text with spaces must be put in quotation marks if specified as an argument.

createSSLCertificate

Creates a default SSL certificate for use in server configuration. Generated keystore file `key.js` is placed under `/resources/security` directory of the server specified in **--server name**. The key algorithm is RSA and signature algorithm is SHA1 with RSA. For more control over the certificate creation, use `keytool` directly.

The arguments are:

--server=name
Specifies the name of the Liberty profile server for keystore creation. This option is required.

--password=passwd
Specifies the password to be used in the keystore, which must be at least 6 characters in length. This option is required.

--validity=days
Specifies the number of days that the certificate is valid, which must be equal to or greater than 365. The default value is 365. This option is optional.

--subject=DN
Specifies the Domain Name (DN) for the certificate subject and issuer. The default value is `CN=localhost,O=ibm,C=us`. This option is optional.

help Prints help information for specified task.

Usage

The following examples demonstrate correct syntax:

```
securityUtility encode GiveMeLiberty
securityUtility createSSLCertificate --server=myserver --password=mypassword --validity=365 --subject=CN=mycompany,
securityUtility help createSSLCertificate
```

Configuring your web application and server for client certificate authentication

You can configure your web application on the Liberty profile using SSL client authentication.

Before you begin

This topic assumes that you have already created the SSL certificates, for example as described in “Creating SSL certificates from the command prompt” on page 21.

About this task

Client certificate authentication occurs if the server side requests that the client side send a certificate. A Websphere server can be configured for client certificate authentication on the SSL configuration. To do this, you add the `ssl-1.0` server feature to the `server.xml` file, along with code that tells the server the keystore information for authentication.

For details of which aspects of SSL are supported, see Liberty profile: Server features.

Procedure

1. Ensure that the deployment descriptor for your web application is specified with`<auth-method>CLIENT-CERT</auth-method>`

   Note: Typically, you would use a tool such as Rational® Application Developer to create the deployment descriptor.
2. Optional: Generate an SSL certificate using the command prompt. See “Liberty profile: securityUtility command” on page 21.
3. Configure your server to enable SSL client authentication by adding the following lines to the `server.xml` file:

```
<featureManager>
     <feature>ssl-1.0</feature>
<featureManager>
```

```
<ssl id="defaultSSLSettings" keyStoreRef="defaultKeyStore"
     trustStoreRef="defaultTrustStore" clientAuthenticationSupported="true" />
<keyStore id="defaultKeyStore" location="key.jks" type="JKS" password="defaultPWD" />
<keyStore id="defaultTrustStore" location="trust.jks" type="JKS" password="defaultPWD" />
```

- If you specify `clientAuthentication="true"`, the server requests that a client send a certificate. However, if the client does not have a certificate, or the certificate is not trusted by the server, the handshake does not succeed.
- If you specify `clientAuthenticationSupported="true"`, the server requests that a client send a certificate. However, if the client does not have a certificate, or the certificate is not trusted by the server, the handshake might still succeed.
- If you do not specify either `clientAuthentication` or `clientAuthenticationSupported`, or you specify `clientAuthentication="false"` or `clientAuthenticationSupported="false"`, the server does not request that a client send a certificate during the handshake.

4. Add a client certificate to your browser. See the documentation of your browser for adding client certificates.
5. Make sure the server trusts any client certificates that are used.
6. Make sure any client certificates used for client authentication are mapped to a user identity in your registry.
   - For the basic registry, the user identity is the common name (CN) from the distinguished name (DN) of the certificate.
   - For a Lightweight Directory Access Protocol (LDAP) registry, the DN from the client certificate must be in the LDAP registry.
7. To fall back to basic authentication (user ID and password only) if client certificate authentication does not succeed, add the following line to your `server.xml` file.

   ```
   <webAppSecurity allowFailOverToBasicAuth="true" />
   ```

   Note: If you specify `allowFailOverToBasicAuth="false"` or do not specify `allowFailOvertoBasicAuth`, and the client certificate authentication does not succeed, the request generates a `403 Authentication error` message and the client is not prompted for basic authentication.

Authenticating users in the Liberty profile

The Liberty profile server uses a user registry to authenticate a user and retrieve information about users and groups to perform security-related operations, including authentication and authorization.

About this task

To learn about how authentication works in the Liberty profile, see Liberty profile: Authentication.

The authentication tasks that you can configure might vary depending on your requirements. Unless you have used the quickStartSecurity element that can configure only one user, you can configure the user registry at the least . You do not have to configure the values for JAAS, authentication Cache and SSO tasks unless you want to change the default values. Configure TAI configuration only when you have an implementation of TAI interface to handle authentication.

You can complete one or more of the following authentication tasks:

Procedure

- Configure authentication cache on the Liberty profile
- Configure a custom JAAS login module for the Liberty profile
- Configure SSO on the Liberty profile
- Configure a user registry for the Liberty profile
- Configure RunAS authentication in the Liberty profile

- Configure TAI for the Liberty profile

Configuring a user registry for the Liberty profile

You can store user and group information for authentication in several types of registry. For example you can use a basic user registry, or an LDAP registry.

Procedure

- Configure a basic user registry for the Liberty profile
- Configure an LDAP user registry for the Liberty profile

Configuring a basic user registry for the Liberty profile

You can configure a basic user registry in the Liberty profile for authentication.

About this task

You can use a basic user registry by defining the users and groups information for authentication on the Liberty profile server. To do this, you add the `appSecurity-1.0` server feature to the `server.xml` file, along with user information in the `basicRegistry` element.

Procedure

1. Add the `appSecurity-1.0` server feature to the `server.xml` file.
2. Optional: To use SSL, add the `ssl-1.0` server feature in the `server.xml` file. See “Enabling SSL communication for the Liberty profile” on page 15.
3. Configure the basic registry for the server as follows:

```
<basicRegistry id="basic" realm="customRealm">
     <user name="mlee" password="p@ssw0rd" />
     <user name="rkumar" password="pa$$w0rd" />
     <user name="gjones" password="{xor}Lz4sLCgwLTs=" />
     <group name="students">
         <member name="mlee" />
         <member name="rkumar" />
     </group>
</basicRegistry>
```

Notes:

- You must use unique names for your users and groups.
- You should remove all trailing and leading spaces from the user and group names.
- If you use the Liberty profile developer tools, the password is encoded for you automatically. If you edit the `server.xml` file directly, you can use the **`securityUtility encode`** command to encode the password for each user. The `securityUtility` command-line tool is available in the `$INSTALL_ROOT/bin` directory. When you run the **`securityUtility encode`** command, you either supply the password to encode as an input from the command line or, if no arguments are specified, the tool prompts you for the password. The tool then outputs the encoded value. Copy the value output by the tool, and use that value for the password. For example, to encode the password `GiveMeLiberty`, run the following command:

  ```
  securityUtility encode GiveMeLiberty
  ```

- A more complete sample configuration of the basic registry is available in file `${wlp.install.dir}/templates/config/basicRegistry.xml`.

Configuring an LDAP user registry with the Liberty profile

You can configure a Lightweight Directory Access Protocol (LDAP) server with the Liberty profile for authentication.

Before you begin

Ensure your LDAP server is up and running, and that the host name and port number of the LDAP server are already in your known list.

About this task

You can use an existing LDAP server for application authentication on the Liberty profile. To do this, you add the `appSecurity-1.0` server feature to the `server.xml` file, and specify in the `server.xml` file the configuration information for connecting to the LDAP server.

Avoid trouble: You can refer to the sample LDAP configuration `ldapRegistry.xml` file in the `${wlp.install.dir}/templates/config` directory, and make sure the configuration in your `server.xml` file is similar to the one in the sample file.

Note: There is no support of certificate filter for LDAP.

Procedure

1. Add the `appSecurity-1.0` server feature to the `server.xml` file.
2. Optional: To communicate with an SSL-enabled LDAP server , add the `ssl-1.0` server feature in the `server.xml` file.
3. Optional: Copy the truststore to the server configuration directory (for example, by using the `${server.config.dir}` variable).

   For SSL communication with an LDAP server to succeed, the Signer certificate for the LDAP server must be added to the truststore that is referenced by the **sslAlias** attribute of the `<ldapRegistry>` element. In the following examples, the Signer certificate must be added to the `LdapSSLTrustStore.jks`.
4. Configure the LDAP entry for the server.

   If you do not need SSL for the LDAP server, remove all SSL and keystore related lines from the following examples.

   You configure the LDAP server in the `server.xml` file or using the Liberty profile developer tools. For sample configuration of other LDAP server, refer to the `${wlp.install.dir}/templates/config/ldapRegistry.xml` file.

   - For IBM Directory Server:

```
<ldapRegistry id="ldap" realm="SampleLdapIDSRealm"
    host="ldapserver.mycity.mycompany.com" port="389" ingnoreCase="true"
    baseDN="o=mycompany,c=us"
    userFilter="(&amp;amp;(uid=%v)(objectclass=ePerson))"
    groupFilter="(&amp;amp;(cn=%v)(|(objectclass=groupOfNames)
                 (objectclass=groupOfUniqueNames)(objectclass=groupOfURLs)))"
    userIdMap="*:uid"
    groupIdMap="*:cn"
    groupMemberIdMap="mycompany-allGroups:member;mycompany-allGroups:uniqueMember;
                      groupOfNames:member;groupOfUniqueNames:uniqueMember"
    ldapType="IBM Tivoli Directory Server"
    sslEnabled="true"
    sslRef="LDAPSSLSettings">
</ldapRegistry>

<sslDefault sslRef="LDAPSSLSettings" />
<ssl id="LDAPSSLSettings" keyStoreRef="LDAPKeyStore" trustStoreRef="LDAPTrustStore" />

<keyStore id="LDAPKeyStore" location="${server.config.dir}/LdapSSLKeyStore.jks"
          type="JKS" password="{xor}CDo9Hgw=" />
<keyStore id="LDAPTrustStore" location="${server.config.dir}/LdapSSLTrustStore.jks"
          type="JKS" password="{xor}CDo9Hgw=" />
```

   - For Microsoft Active Directory Server:

```
<ldapRegistry id="ldap" realm="SampleLdapADRealm"
    host="ldapserver.mycity.mycompany.com" port="389" ignoreCase="true"
    baseDN="cn=users,dc=adtest,dc=mycity,dc=mycompany,dc=com"
    bindDN="cn=testuser,cn=users,dc=adtest,dc=mycity,dc=mycompany,dc=com"
    bindPassword="testuserpwd"
    userFilter="(&amp;(sAMAccountName=%v)(objectcategory=user))"
    groupFilter="(&amp;(cn=%v)(objectcategory=group))"
    userIdMap="user:sAMAccountName"
    groupIdMap="*:cn"
    groupMemberIdMap="memberof:member"
    ldapType="Microsoft Active Directory"
    sslEnabled="true"
    sslRef="LDAPSSLSettings">
</ldapRegistry>

<sslDefault sslRef="LDAPSSLSettings" />
<ssl id="LDAPSSLSettings" keyStoreRef="LDAPKeyStore" trustStoreRef="LDAPTrustStore" />

<keyStore id="LDAPKeyStore" location="${server.config.dir}/LdapSSLKeyStore.jks"
          type="JKS" password="{xor}CDo9Hgw=" />
<keyStore id="LDAPTrustStore" location="${server.config.dir}/LdapSSLTrustStore.jks"
          type="JKS" password="{xor}CDo9Hgw=" />
```

If you use the Liberty profile developer tools, the `bindPassword` password is encoded for you automatically. If you edit the `server.xml` file directly, you can use the **securityUtility encode** command to encode the `bindPassword` password for you. The `securityUtility` command-line tool is available in the `$INSTALL_ROOT/bin` directory. When you run the **securityUtility encode** command, you either supply the password to encode as an input from the command line or, if no arguments are specified, the tool prompts you for the password. The tool then outputs the encoded value. Copy the value output by the tool, and use that value for the `bindPassword` password.

5. Optional: Configure failover for multiple LDAP servers.

```
<ldapRegistry id="LDAP" realm="SampleLdapIDSRealm"
     host="ldapserver1.mycity.mycompany.com" port="389" ignoreCase="true"
      baseDN="o=ibm,c=us" ldapType="IBM Tivoli Directory Server" idsFilters="ibm_dir_server">
 <failoverServers name="failoverLdapServersGroup1">
  <server host="ldapserver2.mycity.mycompany.com" port="389" />
  <server host="ldapserver3.mycity.mycompany.com" port="389" />
 </failoverServers>
 <failoverServers name="failoverLdapServersGroup2">
  <server host="ldapserver4.mycity.mycompany.com" port="389" />
 </failoverServers>
</ldapRegistry>

<idsLdapFilterProperties id="ibm_dir_server"
     userFilter="(&amp;(uid=%v)(objectclass=ePerson))"
     groupFilter="(&amp;(cn=%v)(|(objectclass=groupOfNames)
                 (objectclass=groupOfUniqueNames)(objectclass=groupOfURLs)))"
     userIdMap="*:uid" groupIdMap="*:cn"
     groupMemberIdMap="ibm-allGroups:member;ibm-allGroups:uniqueMember;
                       groupOfNames:member;groupOfUniqueNames:uniqueMember">
</idsLdapFilterProperties>
```

For more information about the `ldapRegistry` and `failoverServers` elements, see Liberty profile: Configuration elements in the server.xml file.

Configuring the authentication cache on the Liberty profile

This topic describes how to modify the way that authenticated users are cached on the Liberty profile.

About this task

Because the creation of a subject is relatively expensive, the Liberty profile provides an authentication cache to store a subject after an authentication of a user is successful. The cache is initialized with a certain number of entries, determined by the **initialSize** attribute, and has a maximum number of entries,

determined by the **maxSize** attribute. If the maximum size is reached, then the least recently used entries are removed from the cache. Also, if a user has been inactive for more than a certain time period determined by the **timeout** attribute, then the entry for that user is removed from the cache. By default, the cache size is initialized to 50 entries and a maximum of 25000 entries with a timeout of 600 seconds.

You do not have to configure the values for the `authCache` element unless you want to change the default values of the authentication cache.

See Authentication Cache for more detail.

Note:

- Any changes to the user registry configuration in `server.xml` file will clear the authentication cache. However, if changes are done to an external user registry (LDAP, for example), the authentication cache is not impacted
- You must consider the following effects of the timeout value on your configuration:
  - Larger authentication cache timeout values can increase the security risk. For example, you might revoke a user in the user registry or repository. However, the revoked user can log in using the credential that is cached in the authentication cache until the cache is refreshed.
  - Smaller authentication cache timeout values can affect performance. When this value is smaller, the Liberty profile server accesses the user registry or repository more frequently.
  - Larger numbers of entries in the authentication cache, which is due to an increased number of users, increases the memory usage by the authentication cache. Thus, the application server might slow down and affect performance.

Procedure

1. Enable the `appSecurity-1.0` server feature in the `server.xml` file.

```
<featureManager>
   <feature>appSecurity-1.0</feature>
</featureManager>
```

2. If you want to change the default options for the authentication cache, add the **authCache** element to the `server.xml` file. In the following example, the initial size of the authentication cache is changed to 100 entries with a maximum of 50000 entries, and the timeout is changed to 15 minutes.

```
<authCache initialSize="100" maxSize="50000" timeout="15m"/>
```

   Note: If you want to disable the authentication cache, set the attribute **cachEnabled** to `false` in the `authentication` element as follows:

```
<authentication id="Basic" cacheEnabled="false" />
```

   For more information on the **authCache** and **authentication** elements, see Liberty profile: Configuration elements in the server.xml file.

Configuring a JAAS custom login module for the Liberty profile

You can configure a custom Java Authentication and Authorization Service (JAAS) login module before or after the Liberty profile server login module.

Before you begin

This topic assumes that you have a JAR file containing the JAAS custom login module, which implements the javax.security.auth.spi.LoginModule interface and uses hashtable, callbacks or shared state variables provided by the Liberty profile server to pass authentication data to the system login module.

About this task

You can use a custom login module to either make additional authentication decisions, or add information to the Subject to make finer-grained authorization decisions inside your application. See JAAS configuration and JAAS login modules for a more detailed overview.

You can also use the developer tools to configure a custom JAAS login module. See "Configuring JAAS on the Liberty profile using developer tools."

See also "Developing JAAS custom login modules for a system login configuration" on page 41.

Procedure

1. Enable the `appSecurity-1.0` server feature in the `server.xml` file.
2. Create a class com.ibm.ws.security.authentication.modules.CustomLoginModule that implements the `LoginModule` interface and package it into the `CustomLoginModule.jar` file.
3. Create a `library` element that uses a `fileset` element indicating where the `CustomLoginModule.jar` file is. In this example, the library**id** is `customLoginLib`.
4. Create a `jaasLoginModule` element. In this example, the **id** is `custom`. Configure the custom login module to require a successful authentication by setting the **controlFlag** attribute to REQUIRED. Set the **libraryRef** attribute to `customLoginLib`, the **id** of the `library` element configured in the previous step. This login module also has two options: **UserRegistry** is `ldap` and **mapToUser** is `user1`.
5. Create a `jaasLogincontextEntry` element with an **id** and **name** of the system-defined JAAS configuration: system_WEB_INBOUND (you can also set this to system.DEFAULT, WSLogin or your own JAAS configuration). On the `loginModuleRef` attribute, add `custom`, the **id** of the `jaasLoginModule` element created in the previous step. Putting this **id** first in the list means that it is the first JAAS login module to be called. You must also list the other default login modules: **hashtable**, **userNameAndPassword**, **certificate** and **token**.

   See the following `server.xml` file as an example:

```
<featureManager>
    <feature>appSecurity-1.0</feature>
</featureManager>

<jaasLoginContextEntry id="system.WEB_INBOUND" name="system.WEB_INBOUND"
                  loginModuleRef="custom, hashtable, userNameAndPassword, certificate, token" />

<jaasLoginModule id="custom"
                  className="com.ibm.ws.security.authentication.modules.CustomLoginModule"
                  controlFlag="REQUIRED" libraryRef="customLoginLib">
    <options userRegistry="ldap" mapToUser="user1"/>
</jaasLoginModule>

<library id="customLoginLib">
    <fileset dir="${server.config.dir}" includes="CustomLoginModule.jar"/>
</library>
...
```

   Note: The option name cannot start with a period (**.**), **config.**, or **service**. Also, the property name **id** or **ID** is not allowed.

   For more information on the `jaasLoginContextEntry`, `jaasLoginModule`, `options` and `library` elements, see Liberty profile: Configuration elements in the server.xml file.

Configuring JAAS on the Liberty profile using developer tools

You can configure a JAAS configuration (system.WEB_INBOUND) with a custom login module for the Liberty profile by editing the configuration. You do not need to configure JAAS unless you want to customize it.

Before you begin

For a description of the underlying process of configuring a server, and detailed information about specific aspects of server configuration, see Configuring the Liberty profile runtime environment.

Avoid trouble: The developer tools creates the reference to a JAAS login module using the `loginModuleRef` element. You must change it and use the **`loginModuleRef`** attribute of `jaasLoginContextEntry` element. You can refer to the sample JAAS configuration `jaasConfig.xml` file in the `${wlp.install.dir}/templates/config` directory, and make sure the configuration in your `server.xml` file is similar to the one in the sample file.

Procedure

1. Select **JAAS Login Context Entry** and click **Add**, then enter the login module names.
2. Select **JAAS Login Module** and configure your custom login module by entering the **Id** and the **Class name**, then click the **New** button and select **Top Level** to enter the Shared Library information.
3. Enter the **ID** for the shared Library in the pop-up panel and click **OK**.
4. Configure the **Name** and **Description** fields for the shared library, then click the **New** button and select **Nested** to add a **Fileset** reference as a nested element.
5. Configure the **Fileset Service Details** by clicking the **Browse** button in the **Base Directory** field and select the directory where the jar file is located. Then, click the **Browse** button in the **Includes pattern** field to select your jar file that contains your custom login module implementation.
6. Optional: If your custom login module needs any options, you can right click **JAAS Login Module**, select **Add** and then select **login module options**.
7. Save the configuration. You can find the following configuration saved in the `server.xml` file.

```
<jaasLoginContextEntry name="system.WEB_INBOUND" id="system.WEB_INBOUND">
    <loginModuleRef>myCustom, hashtable, userNameAndPassword, certificate, token</loginModuleRef>
 </jaasLoginContextEntry>

<jaasLoginModule className="com.ibm.ws.security.authentication.CustomLoginModule"
                 id="myCustom" libraryRef="customLoginLib">
</jaasLoginModule>

<library id="customLoginLib" name="customLoginLib"
         description="Custom login module shared library">
    <fileset dir="${server.config.dir}" includes="CustomLoginModule.jar"/>
</library>
```

8. **Required:** To make the configuration work, you must change the `jaasLoginContextEntry` element to include the **`loginModuleRef`** attribute. You must remove the `loginModuleRef` element and add it as an attribute of the `jaasLoginContextEntry` element.

   Here is an example of configuration using the **`loginModuleRef`** attribute.

```
<jaasLoginContextEntry name="system.WEB_INBOUND" id="system.WEB_INBOUND"
               loginModuleRef="myCustom, hashtable, userNameAndPassword, certificate, token" />

<jaasLoginModule className="com.ibm.ws.security.authentication.CustomLoginModule"
                 id="myCustom" libraryRef="customLoginLib">
</jaasLoginModule>

<library id="customLoginLib" name="customLoginLib"
                            description="Custom login module shared library">
    <fileset dir="${server.config.dir}" includes="CustomLoginModule.jar"/>
</library>
```

Configuring LTPA on the Liberty profile

This topic describes how you can configure a Liberty profile server to use a specific Lightweight Third Party Authentication (LTPA) keys file, user-defined password, and expiration time.

About this task

The LTPA is configured by default when security is enabled for a Liberty profile server for the first time. The default location of the automatically generated LTPA keys file is `${server.config.dir}/resources/security/ltpa.keys`. The keys are encrypted with a random generated key and a default password of `WebAS` is initially used to protect the keys. The password is required when importing the keys into another server. Therefore, to protect the security of the LTPA keys, you must change the password. When the keys are exchanged between the servers, this password must match across the servers for Single Sign On (SSO) to work.

The default expiration timeout is 120 minutes. The expiration value refers to how long the LTPA tokens are valid before they expire.

See LTPA concept in the Liberty profile.

Procedure

1. Configure the `ltpa` element in the `server.xml` file as follows, replacing the sample values in the example with your values.

   ```
   <ltpa keysFileName="yourLTPAKeysFileName.keys" keysPassword="keysPassword" expiration="120" />
   ```

2. Encode the password within the configuration. You can get the encoded value by using the securityUtility encode command.

   For more information on `ltpa` element, see Liberty profile: Configuration elements in the server.xml file.

Customizing SSO configuration using LTPA cookies for the Liberty profile

With single sign-on (SSO) configuration support, web users can authenticate once when accessing Liberty profile resources (such as HTML, JavaServer Pages (JSP) files, and servlets), or accessing resources in multiple Liberty profile servers that share the same Lightweight Third Party Authentication (LTPA) keys.

Example

When a user passes authentication on one of Liberty profile servers, authentication information generated by the server is transported to the browser in a cookie. The cookie is used to propagate the authentication information to other Liberty profile servers.

The LTPA is configured and ready for immediate use. The default cookie name used to store the SSO token is called `ltpaToken2`. If you want to use a different name for the cookie, you can customize the cookie name using the **ssoCookieName** attribute of `webAppSecurity` element. If you customize the cookie name, make sure that all the servers that participate in SSO use the same cookie name.

See SSO concept in the Liberty profile.

The following example code sets the user to be logged out after the HTTP session expires and the name of the SSO cookie as `myCookieName`.

```
<webAppSecurity  logoutOnHttpSessionExpire="true"  ssoCookieName="myCookieName" />
```

Note: In order for SSO to work across servers, the Liberty profile servers must have the same LTPA keys and shared the same user registry.

For details of all the available SSO settings, see the `webAppSecurity` element in Liberty profile: Configuration elements in the server.xml file.

Configuring RunAs authentication in the Liberty profile

You can delegate to another identity during authentication by configuring RunAs specification for the Liberty profile.

About this task

By mapping a specified user identity and optionally password to a RunAs role, you can delegate the authentication process to a user with the RunAs role. You must enable `appSecurity-1.0` server feature and have a user registry for your application to configure the RunAs role.

See RunAs() authentication on how RunAs authentication works.

Procedure

1. Enable `appSecurity-1.0` server feature in the `server.xml` file.
2. Configure a user registry for your application.
3. Specify the run-as element in the deployment descriptor of your application.

   Here is an example of a `web.xml` that specifies subsequent calls be delegated to the user mapped to the role of Employee:

```
<servlet id="Servlet_1">
     <servlet-name>RunAsServlet</servlet-name>
     <display-name>RunAsServlet</display-name>
     <description>RunAsServlet</description>
     <servlet-class>web.RunAsServlet</servlet-class>
     <run-as>
          <role-name>Employee</role-name>
     </run-as>
 </servlet>
```

4. Map this role to a user. You can do this either in the `ibm-application-bnd.xmi/xml` or in the `server.xml` file. In the `run-as` element, you must specify a user name, and you can optionally specify a password. If the password is present, it is recommended to encode it. For example, encode the password using securityUtility encode command in the `/bin` directory of the Liberty profile.

   Here is an example of using `run-as` element within the `application-bnd` element in the `server.xml` file, where the Employee role has been mapped to the RunAs user of **user5** :

```
<application-bnd>
     <security-role name="Employee">
          <user name="user1" />
          <user name="user5" />
          <run-as userid="user5" password="{xor}Lz4sLCgwLTs=" />
     </security-role>
</application-bnd>
```

   Note:

   - Because the password is optional, you can also use the following code for a user without a password:

```
<application-bnd>
      <security-role name="Employee">
           <user name="user1" />
           <user name="user5" />
           <run-as userid="user5" />
       </security-role>
 </application-bnd>
```

   - If you specify the `application-bnd` element in the `server.xml` file, your application must not be in the `dropins` folder. If you leave it in the `dropins` folder, then you must disable application monitoring by setting the following in your `server.xml` file:

```
<applicationMonitor dropinsEnabled="false" />
```

For more information about the run-as element, see Liberty profile: Configuration elements in the server.xml file.

Configuring TAI for the Liberty profile

You can configure the Liberty profile to integrate with a third party security service using Trust Association Interceptors (TAI). The TAI can be called before or after single sign on (SSO).

Before you begin

This topic assumes that you have already installed a third party security server as a reverse proxy server, which can act as a front end authentication server when the Liberty profile server applies its own authorization policy onto the resulting credentials that are passed by the proxy server. Meanwhile, you have a JAR file that contains the custom TAI class, which implements the com.ibm.wsspi.security.tai.TrustAssociationInterceptor interface.

Note: There is no support for monitoring changes of this JAR file.

About this task

A TAI is used to validate HTTP requests between a third party security server and a Liberty profile server. It inspects the HTTP requests from the third party security server to see if there are any security attributes. If the validation for a request is successful in the interceptor, the Liberty profile server authorizes the request by checking whether the client user has the required permission to access the resources.

See also “Developing a custom TAI for the Liberty profile” on page 39 and “Customizing SSO configuration using LTPA cookies for the Liberty profile” on page 30.

You can also use the developer tools to configure a TAI service. See “Configuring TAI on the Liberty profile using developer tools” on page 33

Procedure

1. Enable the appSecurity-1.0 server feature in the server.xml file.
2. Deploy your applications onto the Liberty profile server and enable all required server features, such asjsp-2.2, jdbc-4.0 and so on.
3. Place the TAI implementation library simpleTAI.jar at your server directory.
4. Update the server.xml file with the TAI configuration options and location of the TAI implementation library.

   See the following server.xml file as an example:

```
<featureManager>
    <feature>appSecurity-1.0</feature>
</featureManager>

<trustAssociation id="myTrustAssociation" invokeForUnprotectedURI="false"
                  failOverToAppAuthType="false">
    <interceptors id="simpleTAI" enabled="true"
                  className="com.ibm.websphere.security.sample.SimpleTAI"
                  invokeBeforeSSO="true" invokeAfterSSO="false" libraryRef="simpleTAI">
        <properties hostName="machine1" application="test1"/>

    </interceptors>
</trustAssociation>

<library id="simpleTAI">
    <fileset dir="${server.config.dir}" includes="simpleTAI.jar"/>
</library>
...
```

The custom TAI is enabled in the example, but it does not perform authentication for unprotected URIs and does not allow to fallback to application authentication method if the TAI authentication fails. As shown in the example, the following configuration elements are available for TAI support:

- `trustAssociation`
- `interceptors`
- `properties`

Note: The property name can not start with a period (**.**), **config.**, or **service**. Also, the property name **id** or **ID** is not allowed.

For more information on the `trustAssociation`, `interceptors` and `properties` elements, see also Liberty profile: Configuration elements in the server.xml file.

Configuring TAI on the Liberty profile using developer tools

You can configure a TAI service for the Liberty profile using developer tools.

Before you begin

For a description of the underlying process of configuring a server, and detailed information about specific aspects of server configuration, see Configuring the Liberty profile runtime environment.

Avoid trouble: You can refer to the sample TAI configuration `taiConfig.xml` file in the `${wlp.install.dir}/templates/config` directory, and make sure the configuration in your `server.xml` file is similar to the one in the sample file.

Procedure

1. Select **Trust Association Interceptor Service** and enter an **Id** name.
2. Select **Trust Association Interceptor** and configure the **Id** and the **Class name** which is the fully qualified name of your TAI implementation class, then click the **New** button and select **Top Level** to enter the Shared Library information.
3. Enter the **ID** for the shared Library in the pop-up panel and click **OK**.
4. Configure the **Name** and **Description** fields for the shared library, then click the **New** button and select **Nested** to add a **Fileset** reference as a nested element.
5. Configure the **Fileset Service Details** by clicking the **Browse** button in the **Base Directory** field and select the directory where the jar file is located. Then, click the **Browse** button in the **Includes pattern** field to select your jar file that contains your TAI implementation.
6. Configure **Interceptor properties Details** by clicking the **Add** button to add properties for the interceptor.
7. Save the configuration. You can find the following configuration saved in the `server.xml` file.

```
<trustAssociation id="myTrustAssociation" invokeForUnprotectedURI="false"
                  failOverToAppAuthType="false">
    <interceptors id="simpleTAI" enabled="true"
                  className="com.ibm.websphere.security.sample.SimpleTAI"
                  invokeBeforeSSO="true" invokeAfterSSO="false" libraryRef="simpleTAI">
        <properties hostName="machine1" application="test1"/>
    </interceptors>
</trustAssociation>

<library id="simpleTAI">
    <fileset dir="${server.config.dir}" includes="simpleTAI.jar"/>
</library>
```

Authorizing access to resources in the Liberty profile

The purpose of authorization is to determine whether a user or group has the necessary privileges to access a resource.

About this task

To learn about how authorization works in the Liberty profile, see Liberty profile: Authorization.

The following topics are covered in this section:

Procedure

Configure authorization for applications in a Liberty profile server

Configuring authorization for applications on the Liberty profile

Configuring authorization for your application is to verify whether a user or group belongs to a specified role, and whether this role has the privilege to access a resource.

About this task

The Liberty profile server extracts user and group mapping information from a user registry, then checks an authorization table for the application to determine whether a user or group is assigned to one of the required roles. Then the server reads the deployment descriptor of the application, to determine whether the user or group has the privilege to access the resource.

Procedure

1. Enable the `appSecurity-1.0` server feature in the `server.xml` file.

   For example:

   ```
   <featureManager>
       <feature>appSecurity-1.0</feature>
   </featureManager>
   ```

2. Configure a user registry for authentication on the Liberty profile server.

   See “Authenticating users in the Liberty profile” on page 23.

3. Ensure that the deployment descriptor for your application includes security constraints and other security related information.

   Note: Typically, you would use a tool such as Rational Application Developer to create the deployment descriptor.

4. Configure the authorization information (the user and group to role mapping).

   You can configure the authorization table in the following ways:

   - If you have an EAR file, you can add the authorization table definition to the `ibm-application-bnd.xml` or `ibm-application-bnd.xmi` file.
   - If you have standalone WAR files, you can add the authorization table definitions to the `server.xml` file under the respective application element. You can use the Liberty profile developer tools to do this.

   Notes:

   - If you have an EAR file, the authorization table might already exist. In EAR files that are written to the current specification, this information is stored in an `ibm-application-bnd.xml` file; in older EAR files, this information is stored in an `ibm-application-bnd.xmi` file.
   - If your EAR file does not already contain an `ibm-application-bnd.xm*` file, it is not a straightforward task to create one and you might prefer to add the authorization table to the `server.xml` file.
   - If the authorization table for the EAR file is defined in an `ibm-application-bnd.xm*` file and also in the `server.xml` file, then the two tables are merged. If there are any conflicts, the information from the table in the `server.xml` file is used.

- If you modify your user registry, be sure to review the authorization table for necessary changes. For example, if you are specifying an `access-id` element and change the realm name of the registry, you must also change the realm name in the `access-id` element.
- If you specify the `application-bnd` element in the `server.xml` file, your application must not be in the `dropins` folder. If you leave it in the `dropins` folder, then you must disable application monitoring by setting the following in your `server.xml` file:

  ```
  <applicationMonitor dropinsEnabled="false" />
  ```

A role can be mapped to a user, a group, or a special subject. The two types of special subject are `EVERYONE` and `ALL_AUTHENTICATED_USERS`. When a role is mapped to the `EVERYONE` special subject, there is no security because everyone is allowed access and you are not prompted to enter credentials. When a role is mapped to the `ALL_AUTHENTICATED_USERS` special subject, then any user who has been authenticated by the application server can access the protected resource.

Here is example code for configuring the user and group to role mapping in the `server.xml` file:

```
<application type="war" id="myapp" name="myapp" location="${server.config.dir}/apps/myapp.war">
 <application-bnd>
  <security-role name="user">
   <group name="students" />
  </security-role>
  <security-role name="admin">
   <user name="gjones" />
     <group name="administrators" />
  </security-role>
  <security-role name="AllAuthenticated">
   <special-subject type="ALL_AUTHENTICATED_USERS" />
  </security-role>
 </application-bnd>
</application>
```

In this example, the **admin** role is mapped to the user ID `gjones` and all users in the group `administrators`. The **AllAuthenticatedRole** is mapped to the special subject `ALL_AUTHENTICATED_USERS`, meaning that any user has access as long as they provide valid credentials for authentication.

Accessing JMX connectors on the Liberty profile

This topic describes how to access Java Management Extensions (JMX) connectors on the Liberty profile.

About this task

There are two JMX connectors supported on the Liberty profile, each connector is enabled through a different server feature: `localConnector-1.0` and `restConnector-1.0`.

- The local connector is enabled through the server feature `localConnector-1.0`. Access through the local connector is protected by the policy implemented by the SDK in use. Currently the SDKs require that the client runs on the same host as the Liberty profile, and under the same user ID.
- The REST connector is enabled through the server feature `restConnector-1.0`. Remote access through the REST connector is protected by a single administrator role. In addition, SSL is required to keep the communication confidential. The `restConnector-1.0` feature already includes the `ssl-1.0` feature.

Note: An application deployed on the Liberty profile has unrestricted access to its MBeanServer directory.

The following section describes how to configure and access the REST connector on the Liberty profile.

Procedure

1. Enable the REST connector using the following code in the `server.xml` file.

   ```
   <featureManager>
        <feature>restConnector-1.0</feature>
     </featureManager>
   ```

2. Configure SSL certificates in the server.xml file.
3. Configure a user or group to the administrator role in the server.xml file.
   - Map to the administrator role for the Liberty profile
4. Access the REST connector from a JMX client application or using the jConsole tool provided in the Java SDK. Use -J flags to pass the system properties as Java options and set the class path to include the connector class files. The connector class files are packed in the clients/restConnector.jar file.
   - Use the following properties for SSL certificates:

     ```
     -J-Djavax.net.ssl.trustStore=<location of your client trust store>
     -J-Djavax.net.ssl.trustStorePassword=<password for the trust store>
     -J-Djavax.net.ssl.trustStoreType=<type of trustore>
     ```

     An example of using the jConsole tool with SSL configurations is as follows:

     ```
     jconsole -J-Djava.class.path=%JAVA_HOME%/lib/jconsole.jar;
                                  %JAVA_HOME%/lib/tools.jar;
                                  %WLP_HOME%/clients/restConnector.jar
              -J-Djavax.net.ssl.trustStore=key.jks
              -J-Djavax.net.ssl.trustStorePassword=Liberty
              -J-Djavax.net.ssl.trustStoreType=jks
     ```

     After the jConsole starts, select **Remote Process**, and enter the JMX service URL: service:jmx:rest://<host>:<port>/IBMJMXConnectorREST. You must provide the username and password as well.

Note:

There is no way to pass in timeout options when using thejConsole tool, however, the keys for these options are programmed as system property names in the Liberty profile, so you can specify these options as system properties, where the value passed in the map takes precedence, followed by the system property, and finally the default. For a full list of available options, see Liberty profile: JMX connector options.

Configuring web security related properties for the Liberty profile

You can configure web security related properties for the Liberty profile, such as SSO and client certificate authentication.

About this task

You can use the webAppSecurity element to configure web container application security for the Liberty profile. Make sure you add the appSecurity-1.0 and other required server features to the server.xml file of the Liberty profile.

For all available attributes in the webAppSecurity element, see Liberty profile: Configuration elements in the server.xml file .

You can choose to complete one or more of the following tasks according to your requirements.

Procedure

- “Customizing SSO configuration using LTPA cookies for the Liberty profile” on page 30
- “Configuring your web application and server for client certificate authentication” on page 22

Customizing SSO configuration using LTPA cookies for the Liberty profile

With single sign-on (SSO) configuration support, web users can authenticate once when accessing Liberty profile resources (such as HTML, JavaServer Pages (JSP) files, and servlets), or accessing resources in multiple Liberty profile servers that share the same Lightweight Third Party Authentication (LTPA) keys.

Example

When a user passes authentication on one of Liberty profile servers, authentication information generated by the server is transported to the browser in a cookie. The cookie is used to propagate the authentication information to other Liberty profile servers.

The LTPA is configured and ready for immediate use. The default cookie name used to store the SSO token is called `ltpaToken2`. If you want to use a different name for the cookie, you can customize the cookie name using the **`ssoCookieName`** attribute of `webAppSecurity` element. If you customize the cookie name, make sure that all the servers that participate in SSO use the same cookie name.

See SSO concept in the Liberty profile.

The following example code sets the user to be logged out after the HTTP session expires and the name of the SSO cookie as `myCookieName`.

```
<webAppSecurity  logoutOnHttpSessionExpire="true"  ssoCookieName="myCookieName" />
```

Note: In order for SSO to work across servers, the Liberty profile servers must have the same LTPA keys and shared the same user registry.

For details of all the available SSO settings, see the `webAppSecurity` element in Liberty profile: Configuration elements in the server.xml file.

Configuring your web application and server for client certificate authentication

You can configure your web application on the Liberty profile using SSL client authentication.

Before you begin

This topic assumes that you have already created the SSL certificates, for example as described in “Creating SSL certificates from the command prompt” on page 21.

About this task

Client certificate authentication occurs if the server side requests that the client side send a certificate. A Websphere server can be configured for client certificate authentication on the SSL configuration. To do this, you add the `ssl-1.0` server feature to the `server.xml` file, along with code that tells the server the keystore information for authentication.

For details of which aspects of SSL are supported, see Liberty profile: Server features.

Procedure

1. Ensure that the deployment descriptor for your web application is specified with`<auth-method>CLIENT-CERT</auth-method>`

   Note: Typically, you would use a tool such as Rational Application Developer to create the deployment descriptor.

2. Optional: Generate an SSL certificate using the command prompt. See “Liberty profile: securityUtility command” on page 21.
3. Configure your server to enable SSL client authentication by adding the following lines to the `server.xml` file:

```
<featureManager>
     <feature>ssl-1.0</feature>
<featureManager>

<ssl id="defaultSSLSettings" keyStoreRef="defaultKeyStore"
     trustStoreRef="defaultTrustStore" clientAuthenticationSupported="true" />
<keyStore id="defaultKeyStore" location="key.jks" type="JKS" password="defaultPWD" />
<keyStore id="defaultTrustStore" location="trust.jks" type="JKS" password="defaultPWD" />
```

   - If you specify `clientAuthentication="true"`, the server requests that a client send a certificate. However, if the client does not have a certificate, or the certificate is not trusted by the server, the handshake does not succeed.
   - If you specify `clientAuthenticationSupported="true"`, the server requests that a client send a certificate. However, if the client does not have a certificate, or the certificate is not trusted by the server, the handshake might still succeed.
   - If you do not specify either `clientAuthentication` or `clientAuthenticationSupported`, or you specify `clientAuthentication="false"` or `clientAuthenticationSupported="false"`, the server does not request that a client send a certificate during the handshake.
4. Add a client certificate to your browser. See the documentation of your browser for adding client certificates.
5. Make sure the server trusts any client certificates that are used.
6. Make sure any client certificates used for client authentication are mapped to a user identity in your registry.
   - For the basic registry, the user identity is the common name (CN) from the distinguished name (DN) of the certificate.
   - For a Lightweight Directory Access Protocol (LDAP) registry, the DN from the client certificate must be in the LDAP registry.
7. To fall back to basic authentication (user ID and password only) if client certificate authentication does not succeed, add the following line to your `server.xml` file.

```
<webAppSecurity allowFailOverToBasicAuth="true" />
```

   Note: If you specify `allowFailOverToBasicAuth="false"` or do not specify `allowFailOvertoBasicAuth`, and the client certificate authentication does not succeed, the request generates a `403 Authentication error` message and the client is not prompted for basic authentication.

Configuring JCA security for the Liberty profile

You can configure an authentication data alias to use with a resource reference for Java EE Connector Architecture (JCA) security for authentication on the Liberty profile.

About this task

You can use an authentication data alias by defining a user and password for authentication in the Liberty profile. To do this, add the `jdbc-4.0` server feature to the `server.xml` file and add at least one `authData` element.

Note: There is no Java 2 Connector (J2C) principal mapping module support.

Procedure

1. Add the `jdbc-4.0` server features in the `server.xml` file.

```
<featureManager>
    <feature>jdbc-4.0</feature>
</featureManager>
```

2. Configure the `authData` element in the `server.xml` file as follows. You must use a unique name for the assigned `id` attribute value.

```
<authData id="auth1" user="dbuser1" password="dbuser1pwd"/>
```

3. Configure the IBM deployment descriptor, for example, the `ibm-web-bnd.xml` file, of your application by using the `authentication-alias` element in the resource reference. The `name` attribute value must match the `id` attribute defined in the `server.xml` file.

```
<resource-ref name="jdbc/mydbresource" binding-name="jdbc/mydbresource">
  <authentication-alias name="auth1"/>
</resource-ref>
```

Developing extensions to the Liberty profile security infrastructure

The Liberty profile server provides various plug points so that you can extend the security infrastructure.

About this task

The following topics are covered in this section:

Procedure

- Follow the instructions in “Developing a custom TAI for the Liberty profile” to develop custom trust association interceptors (TAI) to extend the security infrastructure of Liberty profile server.
- Follow the instructions in “Developing JAAS custom login modules for a system login configuration” on page 41 to develop JAAS custom login modules to extend the security infrastructure of Liberty profile server.

Developing a custom TAI for the Liberty profile

You can develop a custom trust association interceptor (TAI) class by implementing the com.ibm.wsspi.security.tai.TrustAssociationInterceptor interface provided in the Liberty profile server.

About this task

The trust association interface is a service provider API that enables the integration of third party security services with a Liberty profile server. When processing the web request, the Liberty profile server calls out and passes the `HttpServletRequest` and `HttpServletResponse` to the trust association interceptors. The `HttpServletRequest` calls the `isTargetInterceptor` method of the interceptor to see whether the interceptor can process the request. After an appropriate trust association interceptor is selected, the `HttpServletRequest` is processed by the `negotiateValidateandEstablishTrust` method of the interceptor, and the result is returned in a `TAIResult` object. You can add your own logic code to each method of the custom TAI class.

See also the Java API document for the TAI interface. The Java API document for each Liberty profile API is detailed in the Programming Interfaces (APIs) section of the information center, and is also available as a JAR file under the `/dev/ibm-api/javadoc` directory of the server image.

Avoid trouble: If you use the developer tools to configure the TAI, refer to the sample TAI configuration `taiConfig.xml` file in the `${wlp.install.dir}/templates/config` directory, and make sure the configuration in your `server.xml` file is similar to the one in the sample file. See “Configuring TAI on the Liberty profile using developer tools” on page 33.

Example

Here is a sample TAI class called `SimpleTAI`, which also lists all available methods from the `TrustAssociationInterceptor` interface.

```
package com.ibm.websphere.security.sample;

import java.util.Properties;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import com.ibm.websphere.security.WebTrustAssociationException;
import com.ibm.websphere.security.WebTrustAssociationFailedException;
import com.ibm.wsspi.security.tai.TAIResult;
import com.ibm.wsspi.security.tai.TrustAssociationInterceptor;

public class SimpleTAI implements TrustAssociationInterceptor {
   public SimpleTAI() {
      super();
   }

   /*
    * @see com.ibm.wsspi.security.tai.TrustAssociationInterceptor#isTargetInterceptor
    * (javax.servlet.http.HttpServletRequest)
    */
   public boolean isTargetInterceptor(HttpServletRequest req)
                  throws WebTrustAssociationException {
      //Add logic to determine whether to intercept this request
      return true;
   }

   /*
    * @see com.ibm.wsspi.security.tai.TrustAssociationInterceptor#negotiateValidateandEstablishTrust
    * (javax.servlet.http.HttpServletRequest,javax.servlet.http.HttpServletResponse)
    */
   public TAIResult negotiateValidateandEstablishTrust(HttpServletRequest req,
                    HttpServletResponse resp) throws WebTrustAssociationFailedException {
        // Add logic to authenticate a request and return a TAI result.
        String tai_user = "taiUser";
        return TAIResult.create(HttpServletResponse.SC_OK, tai_user);
    }

    /*
     * @see com.ibm.wsspi.security.tai.TrustAssociationInterceptor#initialize(java.util.Properties)
     */
    public int initialize(Properties arg0)
                    throws WebTrustAssociationFailedException {
        return 0;
    }

    /*
     * @see com.ibm.wsspi.security.tai.TrustAssociationInterceptor#getVersion()
     */
    public String getVersion() {
        return "1.0";
    }

    /*
     * @see com.ibm.wsspi.security.tai.TrustAssociationInterceptor#getType()
     */
    public String getType() {
        return this.getClass().getName();
```

```
    }

    /*
     * @see com.ibm.wsspi.security.tai.TrustAssociationInterceptor#cleanup()
     */
    public void cleanup()

    {}
}
```

What to do next

Put the custom TAI class in a jar file, for example `simpleTAI.jar`, then make the jar file available to the Liberty profile server. See "Configuring TAI for the Liberty profile" on page 32.

Developing JAAS custom login modules for a system login configuration

For a Liberty profile server, multiple Java Authentication and Authorization Service (JAAS) plug-in points exist for configuring system logins. The Liberty profile uses system login configurations to authenticate incoming requests. You can develop a custom JAAS login module to add information to the **`Subject`** of a system login configuration.

About this task

Application login configurations are called by servlet applications for obtaining a Subject that is based on specific authentication information. When you write a login module that plugs into a Liberty profile application login or system login configuration, you must develop login configuration logic that knows when specific information is present, and how to use the information. See JAAS configuration and JAAS login modules for more details.

Avoid trouble: If you use the developer tools to configure the JAAS custom login module, refer to the sample JAAS configuration `jaasConfig.xml` file in the `${wlp.install.dir}/templates/config` directory, and make sure the configuration in your `server.xml` file is similar to the one in the sample file. See "Configuring JAAS on the Liberty profile using developer tools" on page 28.

To develop a JAAS custom login module for a system login configuration, follow the steps in the procedure:

Procedure

- Understand usable callbacks and how they work.

  See Programmatic login for JAAS for more information about usable callbacks.

  Note: The Liberty profile only supports the following callbacks:

  ```
  callbacks[0] = new javax.security.auth.callback.NameCallback("Username: ");
  callbacks[1] = new javax.security.auth.callback.PasswordCallback("Password: ", false);
  callbacks[2] = new com.ibm.websphere.security.auth.callback.WSCredTokenCallbackImpl("Credential Token: ");
  callbacks[3] = new com.ibm.websphere.security.auth.callback.WSServletRequestCallback("HttpServletRequest: ")
  callbacks[4] = new com.ibm.websphere.security.auth.callback.WSServletResponseCallback("HttpServletResponse: ");
  callbacks[5] = new com.ibm.websphere.security.auth.callback.WSAppContextCallback("ApplicationContextCallback: ");
  callbacks[6] = new WSRealmNameCallbackImpl("Realm Name: ", default_realm);
  callbacks[7] = new WSX509CertificateChainCallback("X509Certificate[]: ");
  callbacks[8] = wsAuthMechOidCallback = new WSAuthMechOidCallbackImpl("AuthMechOid: ");
  ```

- Understand shared state variables and how they work.

If you want to access the objects that the WebSphere Application Server full profile creates during a login, refer to the following shared state variables. For more information about these variables, see the "System Programming Interfaces" subtopic of Programming Interfaces.

com.ibm.wsspi.security.auth.callback.Constants.WSPRINCIPAL_KEY
Specifies an implemented object of the java.security.Principal interface. This shared state variable is for read-only purposes. Do not set this variable in the shared state for custom login modules. The default login module sets this variable.

com.ibm.wsspi.security.auth.callback.Constants.WSCREDENTIAL_KEY
Specifies the com.ibm.websphere.security.cred.WSCredential object. This shared state variable is for read-only purposes. Do not set this variable in the shared state for custom login modules. The default login module will set this variable.

com.ibm.wsspi.security.auth.callback.Constants.WSSSOTOKEN_KEY
Specifies the com.ibm.wsspi.security.token.SingleSignonToken object. Do not set this variable in the shared state for custom login modules. The default login module sets this variable.

- Optional: Understand hashtables for custom JAAS login modules in the Liberty profile. See Hashtable login module for more details.
- Develop a sample custom login module using callbacks and shared state.

  You can use the following sample to learn on how to use some of the callbacks and shared state variables.

```
public class CustomCallbackLoginModule implements LoginModule {

  protected Map<String, ?> _sharedState;
  protected Subject _subject = null;
  protected CallbackHandler _callbackHandler;
  private final String customPrivateCredential = "CustomLoginModuleCredential";

  /**
   * Initialization of login module
   */
  public void initialize(Subject subject, CallbackHandler callbackHandler,
                         Map<String, ?> sharedState, Map<String, ?> options) {
    _sharedState = sharedState;
    _subject = subject;
    _callbackHandler = callbackHandler;
  }

  public boolean login() throws LoginException {
    try {
      AccessController.doPrivileged(new PrivilegedExceptionAction<Object>() {
        public Object run() throws Exception {
          _subject.getPrivateCredentials().add(customPrivateCredential);
          return null;
        }

      });
    } catch (PrivilegedActionException e) {
      throw new LoginException(e.getLocalizedMessage());
    }

    String username = null;
    char passwordChar[] = null;
    byte[] credToken = null;
    HttpServletRequest request = null;
    HttpServletResponse response = null;
    Map appContext = null;
    String realm = null;
    String authMechOid = null;
    java.security.cert.X509Certificate[] certChain = null;
```

```
  NameCallback nameCallback = null;
  PasswordCallback passwordCallback = null;
  WSCredTokenCallbackImpl wsCredTokenCallback = null;
  WSServletRequestCallback wsServletRequestCallback = null;
  WSServletResponseCallback wsServletResponseCallback = null;
  WSAppContextCallback wsAppContextCallback = null;
  WSRealmNameCallbackImpl wsRealmNameCallback = null;
  WSX509CertificateChainCallback wsX509CertificateCallback = null;
  WSAuthMechOidCallbackImpl wsAuthMechOidCallback = null;

  Callback[] callbacks = new Callback[9];
  callbacks[0] = nameCallback = new NameCallback("Username: ");
  callbacks[1] = passwordCallback = new PasswordCallback("Password: ", false);
  callbacks[2] = wsCredTokenCallback = new WSCredTokenCallbackImpl("Credential Token: ");
  callbacks[3] = wsServletRequestCallback = new WSServletRequestCallback("HttpServletRequest: ");
  callbacks[4] = wsServletResponseCallback = new WSServletResponseCallback("HttpServletResponse: ");
  callbacks[5] = wsAppContextCallback = new WSAppContextCallback("ApplicationContextCallback: ");
  callbacks[6] = wsRealmNameCallback = new WSRealmNameCallbackImpl("Realm name:");
  callbacks[7] = wsX509CertificateCallback = new WSX509CertificateChainCallback("X509Certificate[]: ");
  callbacks[8] = wsAuthMechOidCallback = new WSAuthMechOidCallbackImpl("AuthMechOid: ");

  try {
    _callbackHandler.handle(callbacks);
  } catch (Exception e) {
    // handle exception
  }

  if (nameCallback != null)
    username = nameCallback.getName();

  if (passwordCallback != null)
    passwordChar = passwordCallback.getPassword();

  if (wsCredTokenCallback != null)
    credToken = wsCredTokenCallback.getCredToken();

  if (wsServletRequestCallback != null)
    request = wsServletRequestCallback.getHttpServletRequest();

  if (wsServletResponseCallback != null)
    response = wsServletResponseCallback.getHttpServletResponse();

  if (wsAppContextCallback != null)
    appContext = wsAppContextCallback.getContext();

  if (wsRealmNameCallback != null)
    realm = wsRealmNameCallback.getRealmName();

  if (wsX509CertificateCallback != null)
    certChain = wsX509CertificateCallback.getX509CertificateChain();

  if (wsAuthMechOidCallback != null)
    authMechOid = wsAuthMechOidCallback.getAuthMechOid();

  _subject.getPrivateCredentials().add("username = " + username);
  _subject.getPrivateCredentials().add("password = " + String.valueOf(passwordChar));
  _subject.getPrivateCredentials().add("realm = " + realm);
  _subject.getPrivateCredentials().add("authMechOid = " + authMechOid.toString());

  return true;
}

public boolean commit() throws LoginException {
  return true;
```

```
  }

  public boolean abort() {
    return true;
  }

  public boolean logout() {
    return true;
  }

}
```

- Optional: Develop a sample custom login module using hashtable login.

  You can use the following sample to learn on how to use hashtable login.

```
package com.ibm.websphere.security.sample;

import java.util.Map;

import javax.security.auth.Subject;
import javax.security.auth.callback.CallbackHandler;
import javax.security.auth.login.LoginException;
import javax.security.auth.spi.LoginModule;

import com.ibm.wsspi.security.token.AttributeNameConstants;

/**
 * Custom login module that adds another PublicCredential to the subject
 */
@SuppressWarnings("unchecked")
public class CustomHashtableLoginModule implements LoginModule {

    protected Map<String, ?> _sharedState;
    protected Map<String, ?> _options;

    /**
     * Initialization of login module
     */
    public void initialize(Subject subject, CallbackHandler callbackHandler, Map<String, ?> sharedState, Map<String,
        _sharedState = sharedState;
        _options = options;
    }

    public boolean login() throws LoginException {
        try {
            java.util.Hashtable<String, Object> customProperties = (java.util.Hashtable<String, Object>) _sharedStat
            if (customProperties == null) {
                customProperties = new java.util.Hashtable<String, Object>();
            }

            customProperties.put(AttributeNameConstants.WSCREDENTIAL_USERID, "userId");
            // Sample of creating custom cache key
            customProperties.put(AttributeNameConstants.WSCREDENTIAL_CACHE_KEY, "customCacheKey");

            /*
             * Sample for creating user ID and security name
             * customProperties.put(AttributeNameConstants.WSCREDENTIAL_UNIQUEID, "userId");
             * customProperties.put(AttributeNameConstants.WSCREDENTIAL_SECURITYNAME, "securityName");
             * customProperties.put(AttributeNameConstants.WSCREDENTIAL_REALM, "realm");
             * customProperties.put(AttributeNameConstants.WSCREDENTIAL_GROUPS, "groupList");
             */
            /*
             * Sample for creating user ID and password
             * customProperties.put(AttributeNameConstants.WSCREDENTIAL_USERID, "userId");
             * customProperties.put(AttributeNameConstants.WSCREDENTIAL_PASSWORD, "password");
```

```
             */
            Map<String, java.util.Hashtable> mySharedState = (Map<String, java.util.Hashtable>) _sharedState;
            mySharedState.put(AttributeNameConstants.WSCREDENTIAL_PROPERTIES_KEY, customProperties);
        } catch (Exception e) {
            throw new LoginException("LoginException: " + e.getMessage());
        }

        return true;
    }

    public boolean commit() throws LoginException {
        return true;
    }

    public boolean abort() {
        return true;
    }

    public boolean logout() {
        return true;
    }

}
```

What to do next

Add your custom login module into the WEB_INBOUND, and DEFAULT Java Authentication and Authorization Service (JAAS) system login configurations of the `server.xml` file. Put the custom login module class in a JAR file, for example, `customLoginModule.jar`, then make the JAR file available to the Liberty profile server. See “Configuring a JAAS custom login module for the Liberty profile” on page 27.

Customizing an application login to perform an identity assertion using JAAS

Using the Java Authentication and Authorization Service (JAAS) login framework, you can create a JAAS login configuration that can be used to perform login to an identity assertion on the Liberty profile.

About this task

By configuring identity assertion with trust validation, an application can use the JAAS login configuration to perform a programmatic identity assertion. See `IdentityAssertionLoginModule` for more detail.

Avoid trouble: If you use the developer tools to configure the JAAS custom login module, refer to the sample JAAS configuration `jaasConfig.xml` file in the `${wlp.install.dir}/templates/config` directory, and make sure the configuration in your `server.xml` file is similar to the one in the sample file. See “Configuring JAAS on the Liberty profile using developer tools” on page 28.

To customize the application login to perform an identity assertion with trust validation, follow these steps:

Procedure

1. Delegate trust validation to a user implemented plug point. Trust validation must be accomplished in a custom login module. This custom login module should perform any trust validation required, then set the trust and identity information in the shared state to be passed on to the identity assertion login module. A map is required in the shared state key, com.ibm.wsspi.security.common.auth.module.IdentityAssertionLoginModule.state. If the state is missing then a `WSLoginFailedException` is reported by the `IdentityAssertionLoginModule`. This map must include:

- A trust key called com.ibm.wsspi.secuirty.common.auth.module.IdentityAssertionLoginModule.trust. If the key is set to ***true***, then trust is established. If the key is set to ***false***, then no trust is established. If the trust key is not set to true, then the `IdentityAssertionLoginModule` creates a `WSLoginFailedException`.
- An identity key is set: A java.security.Principal can be set in the com.ibm.wsspi.security.common.auth.module.IdentityAssertionLoginModule.principal key.
- Or a java.security.cert.X509Certificate[] can be set in the com.ibm.wsspi.security.common.auth.module.IdentityAssertionLoginModule.certficates key.

If both a principal and certificate are supplied, then the principal is used and a warning is reported.

2. Create a new JAAS configuration for application logins. The JAAS configuration will contain the user implemented trust validation custom login module and the `IdentityAssertionLoginModule`. Then to configure an application login configuration, add the following code in the `server.xml` file:

```
<jaasLoginContextEntry id="CustomIdentityAssertion" name="CustomIdentityAssertion"
                       loginModuleRef="customIdentityAssertion,identityAssertion" />
<jaasLoginModule id="customIdentityAssertion"
                 className="com.ibm.ws.security.authentication.IdentityAssertionLoginModule"
                 controlFlag="REQUIRED" libraryRef="customLoginLib"/>
 <library id="customLoginLib">
    <fileset dir="${server.config.dir}" includes="IdentityAssertionLoginModule.jar"/>
 </library>
```

   This JAAS configuration is then used by the application to perform an Identity Assertion.

3. Perform the programmable identity assertion. A program can now use the JAAS login configuration to perform a programmatic identity assertion. The application program can create a login context for the JAAS configuration created in step 2, then login to that login context with the identity they would assert to. If the login is successful then that identity can be set in the current running process. Here is a example of how such code would operate:

```
NameCallback handler = new NameCallback(new MyPrincipal("Joe"));
LoginContext lc = new LoginContext("customIdentityAssertion", handler);
lc.login();  //assume successful
Subject s = lc.getSubject();
WSSubject.setRunAsSubject(s);
// From here on , the runas identity is "Joe"
```

   Note: The `MyPrincipal` class is the implementation of java.security.Principal interface in the example.

Results

Using the JAAS login framework and two user implemented login modules, you can create a JAAS login configuration that can be used to perform login to an identity assertion.

Chapter 3. How do I secure applications and their environments?

Follow these shortcuts to get started quickly with popular tasks.

When you visit a task in the information center, look for the **IBM Suggests** feature at the bottom of the page. Use it to find available tutorials, demonstrations, presentations, developerWorks® articles, Redbooks®, support documents, and more.

Secure HTTP sessions

Develop applications that use programmatic security

Configure declarative security for EJB applications that use J2EE authorization

Develop programmatic security for EJB applications that use J2EE authorization

Apply Web Services Security (WS-Security) to applications

Enable Java 2 security with the console

Enable Java 2 security with scripting

Developing custom login modules

Enable resource security for J2C and JDBC data sources

Enable resource security for JavaMail

Implement a custom authentication provider using JASPI

Secure the application hosting environment. The counterpart of securing your applications before and after deployment is to secure the server hosting environment into which the applications are deployed.

Secure the administrative environment before installation

Secure the administrative environment after installation

Assign users to roles

Configure security with wsadmin scripting

By default, security is enabled out of box. You have an opportunity to modify the default whenever you create a profile, at installation time or any other time. If you do not deselect it, administrative security will be enabled for a profile. Out of box security authenticates users against the file-based federated repository powered by virtual member manager.

Enable and configure administrative security with the console

Enable and configure administrative security with scripting

Authenticate users with the local operating system user registry

Authenticate users with an LDAP user registry

Authenticate with a custom user registry

Authenticate with the file-based federated repository

Set up single sign-on (SSO)

Access secure resources using SSL and applet clients

Set up Secure Sockets Layer (SSL) between remote servers or clients and servers

Set up CSIv2

Configure an authorization provider

Troubleshoot security

Chapter 4. Task overview: Securing resources

WebSphere Application Server supports the Java Platform, Enterprise Edition (Java EE) model for creating, assembling, securing, and deploying applications. Applications are often created, assembled, and deployed in different phases and by different teams.

About this task

You can secure resources in a Java EE environment by following the required high-level steps. Consult the Java EE specifications for complete details.

Procedure

- Set up and enable security. You must address several issues prior to authenticating users, authorizing access to resources, securing applications, and securing communications. These security issues include migration, interoperability, and installation. After installing WebSphere Application Server, you must determine the proper level of security that is needed for your environment. For more information, see Chapter 5, “Setting up, enabling and migrating security,” on page 51.
- Configure multiple domains. Security domains enable you to define multiple security configurations for use in your environment. For example, you can define different security (such as a different user registry) for user applications than for administrative applications. You can also define separate security configurations for user applications deployed to different servers and clusters. For more information, see Chapter 6, “Configuring multiple security domains,” on page 123
- Authenticate users. The process of authenticating users involves a user registry and an authentication mechanism. Optionally, you can define trust between WebSphere Application Server and a proxy server, configure single sign-on capability, and specify how to propagate security attributes between application servers. For more information, see Chapter 7, “Authenticating users,” on page 159.
- Authorize access to resources. WebSphere Application Server provides many different methods for authorizing accessing resources. For example, you can assign roles to users and configure a built-in or external authorization provider. For more information, see Chapter 8, “Authorizing access to resources,” on page 565.
- Secure communications. WebSphere Application Server provides several methods to secure communication between a server and a client. For more information, see Chapter 9, “Securing communications,” on page 667.
- Develop extensions to the WebSphere security infrastructure. WebSphere Application Server provides various plug points so that you can extend the security infrastructure. For more information, see Chapter 10, “Developing extensions to the WebSphere security infrastructure,” on page 823.
- Use the Auditing Facility to report and track auditable events to ensure the integrity of your system. For more information, see Chapter 11, “Auditing the security infrastructure,” on page 939
- Secure various types of WebSphere applications. See **Securing WebSphere applications** for tasks involving developing, deploying, and administering secure applications, including web applications, web services, and many other types. This section highlights the security concerns and tasks that are specific to each type of application.
- Tune, harden, and maintain security configurations. After you have installed WebSphere Application Server, there are several considerations for tuning, strengthening, and maintaining your security configuration. For more information, see Chapter 12, “Tuning, hardening, and maintaining security configurations,” on page 975.
- Troubleshoot security configurations. For more information, see Chapter 13, “Troubleshooting security configurations,” on page 989.

Results

Your applications and production environment are secured.

Example

See the Security: Resources for learning article for more information on the WebSphere Application Server security architecture.

Chapter 5. Setting up, enabling and migrating security

You must address several issues prior to authenticating users, authorizing access to resources, securing applications, and securing communications. These security issues include migration, interoperability, and installation.

About this task

After installing WebSphere Application Server, you can determine the proper level of security that is needed for your environment. By default, administrative security is enabled and provides the authentication of users using the WebSphere administration functions, the use of Secure Sockets Layer (SSL), and the choice of user account repository.

You can also use the following permissions to enhance security:

- Use the getSSLConfig permission to give your application code the ability to call several of the JSSEHelper methods. For more information about these methods, see the description of the com.ibm.websphere.ssl.JSSEHelper API in the Programming interfaces section of the Information Center.
- Use the AdminPermission permission to give your application code the ability to call WebSphere Application Server administrative APIs. See the topic Setting Java 2 security permissions for an example of how to set this permission.
- Use the accessRuntimeClasses permission to give your application code the ability to load classes that are included with the product. If you are operating in an environment that normally restricts access to these classes, this permission enables your application code to bypass this restriction during class loading. See the topic Global security settings for a description of how to set this permission.

The following information is covered in this section:

Procedure

- Determine if any migration and interoperability issues might affect your installation. For more information, see “Migrating, coexisting, and interoperating – Security considerations.”
- Prepare your environment before and after installing WebSphere Application Server. For more information, see “Preparing for security at installation time” on page 67.
- Enable security for all your application servers or for specific application servers in your realm.

  For more information, see “Enabling security” on page 69.

What to do next

After installing WebSphere Application Server and securing your environment, you must authenticate users. For more information, see Chapter 7, “Authenticating users,” on page 159.

Migrating, coexisting, and interoperating – Security considerations

Use this topic to migrate the security configuration of previous WebSphere Application Server releases and its applications to the new installation of WebSphere Application Server.

Before you begin

This information addresses the need to migrate your security configurations from a previous release of IBM WebSphere Application Server to WebSphere Application Server 8.0. Complete the following steps to migrate your security configurations:

- If security is enabled in the previous release, obtain the administrative server ID and password of the previous release. This information is needed in order to run certain migration jobs.

- You can optionally disable security in the previous release before migrating the installation. No logon is required during the installation.

Note: In WebSphere Application Server Version 8.0, be aware of the following additional migration requirements for security:
  - When migrating from WebSphere Application Server Version 7.x to Version 8.0, if you have a business need to preserve security audit logs from the older release you must first archive the security audit log files in Version 7.x. WebSphere Application Server does not support the migration of security audit log files from the older release to Version 8.0.
  - If your WebSphere Application Server Version 7.x environment is enabled for Kerberos, and you are migrating to version 8.0 on a different machine, the keytab and configuration files for Kerberos must be at the same location on the Version 8.0 machine as on the Version 7.x machine or the configuration will not work.

Procedure

Use the First steps console to access the WebSphere Customization Toolbox, and run the Migration Management Tool.

1. Start the First steps console by launching the `firststeps.bat` or the `firststeps.sh` file. The `firststeps` command is located in the following directory:
   - Linux AIX HP-UX Solaris AIX HP-UX Solaris *./app_server_root/*`profiles/`*`profile_name`*`/firststeps/firststeps.sh`
   - Windows *app_server_root*`\profiles\`*`profile_name`*`\firststeps\firststeps.bat`
2. On the First steps console panel, click **WebSphere Customization Toolbox**.
3. Open the Migration Management Tool.
4. Follow the instructions provided to complete the migration.

Results

The security configuration of previous WebSphere Application Server releases and its applications are migrated to the new installation of WebSphere Application Server Version 8.5.

What to do next

You must migrate any custom class files that are not migrated.

Interoperating with previous product versions

IBM WebSphere Application Server inter-operates with the previous product versions. Use this topic to configure this behavior.

Before you begin

The current release of the Application Server distinguishes the identities of the user who acts as an administrator, managing the Application Server environment, from the identity of the user that is used for authenticating between servers. In prior releases, the end user had to specify a server user ID and password as the user identity for authenticating between servers. In the current release of the Application Server, the server user ID is generated automatically and internally; however, the end user can specify that the server user ID and password not be automatically generated. This option is especially important in the case of a mixed-release cell, where the server user ID and password are specified in a down-level version of the Application Server. In such a scenario, the end user should opt out of automatically generating the server user ID and instead use the server user ID and password that is specified in the down-level version of the Application Server, in order to ensure backwards compatibility.

Interoperability is achieved only when the Lightweight Third Party Authentication (LTPA) authentication mechanism and a distributed user registry is used such as Lightweight Directory Access Protocol (LDAP) or a distributed Custom user registry. LocalOS on most platforms is not considered a distributed user registry (except on z/OS® within the z/OS environment).

Procedure

1. Configure WebSphere Application Server Version 8.5 with the same distributed user registry (that is, LDAP or Custom) that is configured with the previous version. Make sure that the same LDAP user registry is shared by all of the product versions.
   a. In the administrative console, select **Security** > **Global security**.
   b. Choose an available Realm definition and click Configure.
   c. Enter a `Primary administrative user name`. This identity is the user with administrative privileges that is defined in your local operating system. If you are not using the local OS ad the user registry, select the `Server identity that is stored in the user repository`, enter the Server user ID, and the associated password. The user name is used to log on to the administrative console when administrative security is enabled. WebSphere Application Server Version 6.1 requires an administrative user that is distinct from the server user identity so that administrative actions can be audited.

      Attention: In WebSphere Application Server, Version 6.0.x, a single user identity is required for both administrative access and internal process communication. When migrating to Version 8.5, this identity is used as the server user identity. You need to specify another user for the administrative user identity.
   d. When interoperating with Version 6.0.x or previous versions, you must select the Server identity that is stored in the user repository. Enter the `Server user id` and the associated `Password`.
2. Configure the LTPA authentication mechanism. Automatic generation of the LTPA keys should be disabled. If not, keys used by a previous release are lost. Export the current LTPA keys from WebSphere Application Server Version 8.0 and import them into the previous release or export the LTPA keys from the previous release into Version 8.0.
   a. In the administrative console select **Security** > **Global security**.
   b. From Authentication mechanisms and expiration, click LTPA.
   c. Click the Key set groups link , then click the key set group that displays in the Key set groups panel.
   d. Clear the Automatically generate keys check box.
   e. Click OK, then click Authentication mechanisms and expiration in the path at the top of the Key set groups panel.
   f. Scroll down to the Cross-cell single sign-on section, and enter a password to use for encrypting the LTPA keys when adding them to the file.
   g. Enter the password again to confirm the password.
   h. Enter the Fully qualified key file name that contains the exported keys.
   i. Click Export keys.
   j. Follow the instructions provided in the previous release to import the exported LTPA keys into that configuration.
3. If you are using the default SSL configuration, extract all of the signer certificates from the WebSphere Application Server Version 8.5 common trust store. Otherwise, extract signers where necessary to import them into the previous release.
   a. In the administrative console, click **Security** > **SSL certificate and key management**.
   b. Click Key stores and certificates.
   c. Click NodeDefaultTrustStore.
   d. Click Signer certificates.
   e. Select one signer and click Extract.

f. Enter a unique path and filename for the signer. For example, `/tmp/signer1.arm`.

g. Click OK. Repeat for all of the signers in the trust store.

h. Check other trust stores for other signers that might need to be shared with the other server. Repeat steps e through h to extract the other signers.

You can also import a signer certificate, which is also called a certificate authority (CA) certificate, from a truststore on a non-z/OS platform server to a z/OS keyring. the z/OS keyring contains the signer certificates that originated on the non-z/OS platform server. For more information, see Importing a signer certificate from a truststore to a z/OS keyring.

4. Add the exported signers to `DummyServerTrustFile.jks` and `DummyClientTrustFile.jks` in the `/etc` directory of the back-level product version. If the previous release is not using the dummy certificate, the signer certificate(s) from the previous release must be extracted and added into the WebSphere Application Server Version 8.5 release to enable SSL connectivity in both directions.
   a. Open the key management utility, iKeyman, for that product version.
   b. Start `ikeyman.bat` or `ikeyman.sh` from the `${USER_INSTALL_ROOT}/bin` directory.
   c. Select **Key Database File** > **Open**.
   d. Open `${USER_INSTALL_ROOT}/etc/DummyServerTrustFile.jks`.
   e. Enter WebAS for the password.
   f. Select Add and enter one of the files extracted in step 2. Continue until you have added all of the signers.
   g. Repeat steps c through f for the `DummyClientTrustFile.jks` file.
5. Verify that the application uses the correct Java Naming and Directory Interface (JNDI) name and naming bootstrap port for performing a naming lookup.
6. Stop and restart all of the servers.

Interoperating with a C++ common object request broker architecture client

WebSphere Application Server supports security in the CORBA C++ client to access-protected enterprise beans. If configured, C++ CORBA clients can access protected enterprise bean methods using a client certificate to achieve mutual authentication on WebSphere Application Server applications.

About this task

You can achieve interoperability of Security Authentication Service between the C++ Common Object Request Broker Architecture (CORBA) client and WebSphere Application Server using Common Secure Interoperability Version 2 (CSIv2) authentication protocol over Remote Method Invocation over the Internet Inter-ORB Protocol (RMI-IIOP). The CSIv2 security service protocol has authentication, attribute and transport layers. Among the three layers, transport authentication is conceptually simple, however, cryptographically based transport authentication is the strongest. WebSphere Application Server has implemented the transport authentication layer, so that C++ secure CORBA clients can use it effectively in making CORBA clients and protected enterprise bean resources work together.

Security authentication from non-Java based C++ client to enterprise beans. WebSphere Application Server supports security in the CORBA C++ client to access-protected enterprise beans. If configured, C++ CORBA clients can access protected enterprise bean methods using a client certificate to achieve mutual authentication on WebSphere Application Server applications.

To support the C++ CORBA client in accessing protected enterprise beans, complete the following steps:

- Create an environment file for the client, such as `current.env`. Set the variables presented in the following list in the file:

Table 7. Environment Variables.

This table lists the environment variables needed to support the C++ CORBA client in accessing protected enterprise beans.

C++ security setting	Description
client_protocol_password	Specifies the password for the user ID.
client_protocol_user	Specifies the user ID to authenticate at the target server.
security_sslKeyring	Specifies the name of the RACF keyring for the client to use. The keyring must be defined under the user ID that is issuing the command to run the client.

- Point to the environment file using the fully qualified path name through the WAS_CONFIG_FILE environment variable. For example, in the `test.sh` test shell script, export:

  ```
  /WebSphere/V6R0M0/DeploymentManager/profiles/default/config/cells
    /PLEX1Network/nodes/PLEX1Manager/servers/dmgr
  ```

  Some of the environment file terms are explained below:

  default
  profile name

  PLEX1Network
  cell name

  PLEX1Manager
  node name

  dmgr server name

Procedure

1. Obtain a valid certificate to represent the client and export its public key to the target enterprise bean server.

   A valid certificate is needed to represent the C++ client. Request a certificate from the certificate authority (CA) or create a self-signed certificate for testing purposes.

   Use the Key Management Utility from the Global Security Kit (GSKit) to extract the public key from the personal certificate and save it in the `.arm` format.
2. Prepare a truststore file for WebSphere Application Server.

   Add the extracted client public key in the `.arm` file from the client to the server key truststore file. The server can now authenticate the client.

   Note: This is done by invoking the Key Management Utility through `ikeyman.bat` or `ikeyman.sh` from WebSphere Application Server installation.
3. Configure WebSphere Application Server to support Secure Sockets Layer (SSL) as the authentication mechanism.
   a. Start the administrative console.
   b. Locate the application server that has the target enterprise bean deployed and configure it to use SSL client certificate authentication.

      If it is a base installation, complete the following steps:

      1) Click **Security > Global security**. Under RMI/IIOP security, click **CSIv2 inbound communications**. Select **Supported** for the Basic authentication and Client certificate authentication options. Leave the rest of the options as defaults.
      2) Click **OK**.
      3) Click **Security > Global security**. Under RMI/IIOP security, click **CSIv2 inbound communications** and verify that the **SSL-supported** option is selected under Transport.

      If it is a WebSphere Application Server, Network Deployment setting, complete the following steps:

1) Click **Servers > Application Servers >** *server_name_where_the_EJB_resides*.
2) Under Security, click **Server security**.
3) Select the **RMI/IIOP security for this server overrides cell settings** option.
4) Under Additional properties, click **CSIv2 inbound communications**.
5) Select **Supported** for the Basic authentication and Client certificate authentication options. Leave the rest of the options as defaults.
6) Click **Servers > Application Servers >** *server_name_where_the_EJB_resides*.
7) Under Security, click **Server security**.
8) Under Additional properties, click **CSIv2 inbound communications**.
9) Verify that the **SSL-Supported** option is selected.

c. Restart the application server.

The WebSphere Application Server is ready to take a C++ CORBA security client and a mutually authenticated server and client by using SSL in the transport layer.

4. Configure the C++ CORBA client to use a certificate in performing the mutual authentication.

Client users are accustomed to using property files in their applications because they are helpful in specifying configuration settings. The following list presents important C++ security settings:

Table 8. C++ security properties.

This tables lists important C++ security settings.

C++ security setting	Description
com.ibm.CORBA.bootstrapHostName=ricebella.austin.ibm.com	Specifies the target host name.
com.ibm.CORBA.securityEnabled=yes	Enables security.
com.ibm.CSI.performTLClientAuthenticationSupported=yes	Ensures client is supporting mutual authentication by certificate
com.ibm.ssl.keyFile=C:/ricebella/etc/DummyKeyRingFile.KDB	Specifies which key database file to use.
com.ibm.ssl.keyPassword=WebAS	Specifies the password for opening the key database file. WebSphere Application Server supports a utility called `PasswordEncode4cpp` to encode the plain password.
com.ibm.CORBA.translationEnabled=1	Enables the valueType conversion.

To use the property files in running a C++ client, an environment variable WASPROPS, is used to indicate where a property file or a list of property files exists.

For the complete set of C++ client properties, see the sample property file `scclient.props`, which is shipped with the product located in the *app_server_root*`/profiles/`*profile_name*`/etc` directory.

Migrating trust association interceptors

Use this topic to manually migrate trust associations.

Before you begin

Note: Data sources are not supported for use within a Trust Association Interceptor (TAI). Data sources are intended for use within J2EE applications and designed to operate within the EJB and web containers. Trust Association Interceptors do not run within a container, and while data sources may function in the TAI environment, they are untested and not guaranteed to function properly.

The following topics are addressed in this document:
- Changes to the product-provided trust association interceptors
- Migrating product-provided trust association interceptors
- Changes to the custom trust association interceptors
- Migrating custom trust association interceptors

Changes to the product-provided trust association interceptors

For the product-provided implementation for the WebSEAL server, a new optional com.ibm.websphere.security.webseal.ignoreProxy property is added. If this property is set to `true` or `yes`, the implementation does not check for the proxy host names and the proxy ports to match any of the host names and ports that are listed in the com.ibm.websphere.security.webseal.hostnames and the com.ibm.websphere.security.webseal.ports property respectively. For example, if the VIA header contains the following information:

```
HTTP/1.1 Fred (Proxy), 1.1 Sam (Apache/1.1),
HTTP/1.1 webseal1:7002, 1.1 webseal2:7001
```

and the com.ibm.websphere.security.webseal.ignoreProxy property is set to `true` or `yes`, the host name `Fred`, is not used when matching the host names. By default, this property is not set, which implies that any proxy host names and ports that are expected in the VIA header are listed in the host names and the ports properties to satisfy the isTargetInterceptor method.

The previous VIA header information was split onto two lines for illustrative purposes only.

For more information about the com.ibm.websphere.security.webseal.ignoreProxy property, see the article in the information center on configuring single signon using trust association interceptor ++.

Migrating product-provided trust association interceptors

The properties that are located in the `webseal.properties` and `trustedserver.properties` files are not migrated from previous versions of WebSphere Application Server. You must migrate the appropriate properties to WebSphere Application Server Version 6.0.*x* using the trust association panels in the administrative console. For more information, see Configuring trust association interceptors.

Changes to the custom trust association interceptors

If the custom interceptor extends the com.ibm.websphere.security.WebSphereBaseTrustAssociationInterceptor property, implement the following new method to initialize the interceptor:

```
public int init (java.util.Properties props);
```

WebSphere Application Server checks the return status before using the trust association implementation. Zero (0) is the default value for indicating that the interceptor is successfully initialized.

However, if a previous implementation of the trust association interceptor returns a different error status, you can either change your implementation to match the expectations or make one of the following changes:

Method 1:

Add the com.ibm.websphere.security.trustassociation.initStatus property in the trust association interceptor custom properties. Set the property to the value that indicates the interceptor is successfully initialized. All of the other possible values imply failure. In case of failure, the corresponding trust association interceptor is not used.

Method 2:

Add the com.ibm.websphere.security.trustassociation.ignoreInitStatus property in the trust association interceptor custom properties. Set the value of this property to `true`, which tells WebSphere Application Server to ignore the status of this method. If you add this property to the custom properties, WebSphere Application Server does not check the return status, which is similar to previous versions of WebSphere Application Server.

The public int init (java.util.Properties props method replaces the public int init (String propsFile) method.

The init(Properties) method accepts a java.util.Properties object, which contains the set of properties that is required to initialize the interceptor. All of the properties set for an interceptor are sent to this method.

The interceptor can then use these properties to initialize itself. For example, in the product-provided implementation for the WebSEAL server, this method reads the hosts and ports so that a request coming in can be verified to come from trusted hosts and ports. A return value of Zero (0) implies that the interceptor initialization is successful. Any other value implies that the initialization is not successful and the interceptor is not used.

The init(String) method still works if you want to use it instead of implementing the init(Properties) method. The only requirement is that you enter the file name containing the custom trust association properties using the **Custom Properties** link of the interceptor in the administrative console or by using scripts. You can enter the property using either of the following methods. The first method is used for backward compatibility with previous versions of WebSphere Application Server.

Method 1:

The same property names used in the previous release are used to obtain the file name. The file name is obtained by concatenating `.config` to the com.ibm.websphere.security.trustassociation.types property value. If the `myTAI.properties` file is located in the *app_server_root*`/properties` directory, set the following properties:

- `com.ibm.websphere.security.trustassociation.types = myTAItype`
- `com.ibm.websphere.security.trustassociation.myTAItype.config =` *app_server_root*`/properties/myTAI.properties`

Method 2:

You can set the com.ibm.websphere.security.trustassociation.initPropsFile property in the trust association custom properties to the location of the file. For example, set the following property:

```
com.ibm.websphere.security.trustassociation.initPropsFile=
app_server_root/properties/myTAI.properties
```

The previous line of code is split into two lines for illustrative purposes only. Type as one continuous line.

However, it is highly recommended that your implementation be changed to implement the init(Properties) method instead of relying on the init (String propsfile) method.

Migrating custom trust association interceptors

The trust associations from previous versions of WebSphere Application Server are not automatically migrated to WebSphere Application Server Version 8.5. You can manually migrate these trust associations using the following steps:

Procedure

1. Recompile the implementation file, if necessary.

   For more information, refer to the "Changes to the custom trust association interceptors" section previously discussed in this document.

   To recompile the implementation file, type the following code:

```
%WAS_HOME%/java/bin/javac -classpath %WAS_HOME%/plugins/com.ibm.ws.runtime.jar;
%WAS_HOME%/dev/JavaEE/j2ee.jar your_implementation_file.java
```

   The previous line of code is broken into two lines for illustrative purposes only. Type the code as one continuous line.

2. Identify the trust association interceptor class file for use when the server is restarted. Place the file either at the*app_server_root/classes* directory OR use the Java Virtual Machine (JVM) system property, `-Dws.ext.dirs` to specify where the file resides.
3. Restart all the serversWebSphere Application Server.
4. Enable security to use the trust association interceptor. The properties that are located in your custom trust association properties file and in the `trustedserver.properties` file are not migrated from

previous versions of WebSphere Application Server. You must migrate the appropriate properties to WebSphere Application Server Version 8.5 using the trust association panels in the administrative console.

For more information, see Configuring trust association interceptors.

Migrating Common Object Request Broker Architecture programmatic login to Java Authentication and Authorization Service (CORBA and JAAS)

Use this topic as an example of how to perform programmatic login using the CORBA-based programmatic login APIs.

Before you begin

This document outlines the deprecated Common Object Request Broker Architecture (CORBA) programmatic login APIs and the alternatives that are provided by JAAS. WebSphere Application Server fully supports the Java Authentication and Authorization Service (JAAS) as programmatic login application programming interfaces (API). Refer to the *Securing applications and their environment* PDF for more details on JAAS support.

The following list includes the deprecated CORBA programmatic login APIs.

- `${user.install.root}/installedApps/sampleApp.ear/default_app.war/WEB-INF/classes/LoginHelper.java`.
- `${user.install.root}/installedApps/sampleApp.ear/default_app.war/WEB-INF/classes/ServerSideAuthenticator.java`.
- **org.omg.SecurityLevel2.Credentials**. This API is included with the product, but it is not recommended that you use the API.

The APIs that are provided in WebSphere Application Server are a combination of standard JAAS APIs and a product implementation of standard JAAS interfaces.

The following information is only a summary; refer to the JAAS documentation for your platform located at: http://www.ibm.com/developerworks/java/jdk/security/ .

- Programmatic login APIs:
  - javax.security.auth.login.LoginContext
  - javax.security.auth.callback.CallbackHandler interface: The WebSphere Application Server product provides the following implementation of the javax.security.auth.callback.CallbackHandler interface:
    com.ibm.websphere.security.auth.callback.WSCallbackHandlerImpl
    Provides a non-prompt CallbackHandler handler when the application pushes basic authentication data (user ID, password, and security realm) or token data to product login modules. This API is recommended for server-side login.
    com.ibm.websphere.security.auth.callback.WSGUICallbackHandlerImpl
    Provides a login prompt CallbackHandler handler to gather basic authentication data (user ID, password, and security realm). This API is recommended for client-side login.

    If this API is used on the server side, the server is blocked for input.
  - javax.security.auth.callback.Callback interface:
    javax.security.auth.callback.NameCallback
    Provided by JAAS to pass the user name to the LoginModules interface.
    javax.security.auth.callback.PasswordCallback
    Provided by JAAS to pass the password to the LoginModules interface.
    com.ibm.websphere.security.auth.callback.WSCredTokenCallbackImpl
    Provided by the product to perform a token-based login. With this API, an application can pass a token-byte array to the LoginModules interface.
  - **javax.security.auth.spi.LoginModule interface**

WebSphere Application Server provides a LoginModules implementation for client and server-side login. Refer to the *Securing applications and their environment* PDF for details.

- javax.security.Subject:

  com.ibm.websphere.security.auth.WSSubject
  An extension provided by the product to invoke remote J2EE resources using the credentials in the javax.security.Subject

  com.ibm.websphere.security.cred.WSCredential
  After a successful JAAS login with the WebSphere Application Server LoginModules interfaces, a com.ibm.websphere.security.cred.WSCredential credential is created and stored in the Subject.

  com.ibm.websphere.security.auth.WSPrincipal
  An authenticated principal that is created and stored in a Subject that is authenticated by the WebSphere Application Server LoginModules interface.

Procedure

1. Use the following as an example of how to perform programmatic login using the CORBA-based programmatic login APIs: The CORBA-based programmatic login APIs are replaced by JAAS login.

   Note: The LoginHelper application programming interface (API) that is used in the following example is deprecated in WebSphere Application Server Version 8.5 and will be removed in a future release. It is recommended that you use the JAAS programmatic login APIs that are shown in the next step.

```
public class TestClient {
...
private void performLogin() {
// Get the ID and password of the user.
String userid = customGetUserid();
String password = customGetPassword();

// Create a new security context to hold authentication data.
LoginHelper loginHelper = new LoginHelper();
try {
// Provide the ID and password of the user for authentication.
org.omg.SecurityLevel2.Credentials credentials =
loginHelper.login(userid, password);

// Use the new credentials for all future invocations.
loginHelper.setInvocationCredentials(credentials);
// Retrieve the name of the user from the credentials
// so we can tell the user that login succeeded.

String username = loginHelper.getUserName(credentials);
System.out.println("Security context set for user: "+username);
} catch (org.omg.SecurityLevel2.LoginFailed e) {
// Handle the LoginFailed exception.
}
}
...
}
```

2. Use the following example to migrate the CORBA-based programmatic login APIs to the JAAS programmatic login APIs.

   The following example assumes that the application code is granted for the required Java 2 security permissions. For more information, see the *Securing applications and their environment* PDF and the JAAS documentation located at http://www.ibm.com/developerworks/java/jdk/security/.

```
public class TestClient {
...
private void performLogin() {
// Create a new JAAS LoginContext.
javax.security.auth.login.LoginContext lc = null;

try {
// Use GUI prompt to gather the BasicAuth data.
lc = new javax.security.auth.login.LoginContext("WSLogin",
new com.ibm.websphere.security.auth.callback.WSGUICallbackHandlerImpl());
```

```
// create a LoginContext and specify a CallbackHandler implementation
// CallbackHandler implementation determine how authentication data is collected
// in this case, the authentication date is collected by  login prompt
//   and pass to the authentication mechanism implemented by the LoginModule.
} catch (javax.security.auth.login.LoginException e) {
System.err.println("ERROR: failed to instantiate a LoginContext and the exception: "
+ e.getMessage());
e.printStackTrace();

// may be javax.security.auth.AuthPermission "createLoginContext" is not granted
//   to the application, or the JAAS Login Configuration is not defined.
}

if (lc != null)
try {
lc.login();  // perform login
javax.security.auth.Subject s = lc.getSubject();
// get the authenticated subject

// Invoke a J2EE resources using the authenticated subject
com.ibm.websphere.security.auth.WSSubject.doAs(s,
new java.security.PrivilegedAction() {
public Object run() {
try {
bankAccount.deposit(100.00);  // where bankAccount is an protected EJB
} catch (Exception e) {
System.out.println("ERROR: error while accessing EJB resource, exception: "
+ e.getMessage());
e.printStackTrace();
}
return null;
}
}
);

// Retrieve the name of the principal from the Subject
// so we can tell the user that login succeeded,
// should only be one WSPrincipal.
java.util.Set ps =
s.getPrincipals(com.ibm.websphere.security.auth.WSPrincipal.class);
java.util.Iterator it = ps.iterator();
while (it.hasNext()) {
com.ibm.websphere.security.auth.WSPrincipal p =
(com.ibm.websphere.security.auth.WSPrincipal) it.next();
System.out.println("Principal: " + p.getName());
}
} catch (javax.security.auth.login.LoginException e) {
System.err.println("ERROR: login failed with exception: " + e.getMessage());
e.printStackTrace();

// login failed, might want to provide relogin logic
}
}
...
}
```

Migrating from the CustomLoginServlet class to servlet filters

Use this topic to allow migration in an application that uses form-based login and servlet filters without the use of the CustomLoginServlet class.

Before you begin

The CustomLoginServlet class is deprecated in WebSphere Application Server Version 5. Those applications using the CustomLoginServlet class to perform authentication now need to use form-based login. Using the form-based login mechanism, you can control the look and feel of the login screen. In form-based login, a login page is specified and displays when retrieving the user ID and password information. You also can specify an error page that displays when authentication fails.

If login and error pages are not enough to implement the CustomLoginServlet class, use servlet filters. Servlet filters can dynamically intercept requests and responses to transform or use the information that is contained in the requests or responses. One or more servlet filters attach to a servlet or a group of servlets. Servlet filters also can attach to JavaServer Pages (JSP) files and HTML pages. All the attached servlet filters are called before invoking the servlet.

Both form-based login and servlet filters are supported by any Servlet 2.3 specification-compliant web container. A form login servlet performs the authentication and servlet filters can perform additional authentication, auditing, or logging tasks.

To perform pre-login and post-login actions using servlet filters, configure these servlet filters for either form login page or for /j_security_check URL. The j_security_check is posted by the form login page with the j_username parameter that contains the user name and the j_password parameter that contains the password. A servlet filter can use user name and password information to perform more authentication or meet other special needs.

Procedure

1. Develop a form login page and error page for the application.
   Refer to the *Securing applications and their environment* PDF for details.
2. Configure the form login page and the error page for the application.
   Refer to the *Securing applications and their environment* PDF for details.
3. Develop servlet filters if additional processing is required before and after form login authentication.
   Refer to the *Securing applications and their environment* PDF for details.
4. Configure the servlet filters that are developed in the previous step for either the form login page URL or for the /j_security_check URL. Use an assembly tool or development tools like Rational Application Developer to configure filters. After configuring the servlet filters, the `web-xml` file contains two stanzas. The first stanza contains the servlet filter configuration, the servlet filter, and its implementation class. The second stanza contains the filter mapping section and a mapping of the servlet filter to the URL.
   For more information, see the *Securing applications and their environment* PDF.

Results

This migration results in an application that uses form-based login and servlet filters without the use of the CustomLoginServlet class.

What to do next

The new application uses form-based login and servlet filters to replace the CustomLoginServlet class. Servlet filters also are used to perform additional authentication, auditing, and logging.

Migrating Java 2 security policy

Use this topic for guidance pertaining to migrating Java 2 security policy.

About this task

Previous WebSphere Application Server releases

WebSphere Application Server uses the Java 2 security manager in the server runtime to prevent enterprise applications from calling the System.exit and the System.setSecurityManager methods. These two Java application programming interfaces (API) have undesirable consequences if called by enterprise applications. The System.exit API, for example, causes the Java virtual machine (application server process) to exit prematurely, which is not a beneficial operation for an application server.

To support Java 2 security properly, all the server runtime must be marked as `privileged` (with doPrivileged API calls inserted in the correct places), and identify the default permission sets or policy. Application code is not privileged and subject to the permissions that are defined in the policy files. The doPrivileged instrumentation is important and necessary to support Java 2 security. Without it, the application code must be granted the permissions that are required by the server runtime. This situation is due to the design and algorithm that is used by Java 2 security to enforce permission checks. Refer to the Java 2 security check permission algorithm.

The following two permissions are enforced by the Java 2 security manager (hard coded) for WebSphere Application Server:
- java.lang.RuntimePermission(exitVM)
- java.lang.RuntimePermission(setSecurityManager)

Application code is denied access to these permissions regardless of what is in the Java 2 security policy. However, the server runtime is granted these permissions. All the other permission checks are not enforced.

Only two permissions are supported:
- java.net.SocketPermission
- java.net.NetPermission

However, not all the product server runtime is properly marked as privileged. You must grant the application code all the other permissions besides the two listed previously or the enterprise application can potentially fail to run. This Java 2 security policy for enterprise applications is liberal.

What changed

Java 2 Security is fully supported in WebSphere Application Server, which means that all permissions are enforced. The default Java 2 security policy for an enterprise application is the recommended permission set defined by the Java Platform, Enterprise Edition (Java EE) Version 1.4 specification. Refer to the *profile_root*`/config/cells/`*cell_name*`/nodes/`*node_name*`/app.policy` file for the default Java 2 security policy that is granted to enterprise applications. This policy is a much more stringent compared to previous releases.

All policy is declarative. The product security manager honors all policy that is declared in the policy files. There is an exception to this rule: enterprise applications are denied access to permissions that are declared in the *profile_root*`/config/cells/`*cell_name*`/filter.policy` file.

Note: The default Java 2 security policy for enterprise applications is much more stringent and all the permissions are enforced in WebSphere Application Server Version 8.5. The security policy might fail because the application code does not have the necessary permissions granted where system resources, such as file I/O, can be programmatically accessed and are now subject to the permission checking.

In application code, do not use the setSecurityManager permission to set a security manager. When an application uses the setSecurityManager permission, there is a conflict with the internal security manager within WebSphere Application Server. If you must set a security manager in an application for RMI purposes, you also must enable the **Use Java 2 security to restrict application access to local resources** option on the Global security page within the WebSphere Application Server administrative console. WebSphere Application Server then registers a security manager. The application code can verify that this security manager is registered by using System.getSecurityManager() application programming interface (API).

Migrating system properties

The following system properties are used in previous releases in relation to Java 2 security:

- **java.security.policy**. The absolute path of the policy file (action required). This system property contains both system permissions (permissions granted to the Java virtual machine (JVM) and the product server runtime) and enterprise application permissions. Migrate the Java 2 security policy of the enterprise application to Version 8.5. For Java 2 security policy migration, see the steps for migrating Java 2 security policy.
- **enableJava2Security**. Used to enable Java 2 security enforcement (no action required). This system property is deprecated; a flag in the WebSphere configuration application programming interface (API) is used to control whether to enable Java 2 security. Enable this option through the administrative console.
- **was.home**. Expanded to the installation directory of WebSphere Application Server (action might be required). This system property is deprecated; superseded by the ${user.install.root} and ${was.install.root} properties. If the directory contains instance-specific data then ${user.install.root} is used; otherwise ${was.install.root} is used. Use these properties interchangeably for the WebSphere Application Server or the WebSphere Application Server, Network Deployment environments. See the steps for migrating Java 2 security policy.

Migrating the Java 2 Security Policy

No easy way exists to migrate the Java policy file to Version 8.5 automatically because of a mixture of system permissions and application permissions in the same policy file. Manually copy the Java 2 security policy for enterprise applications to a `was.policy` or `app.policy` file. However, migrating the Java 2 security policy to a `was.policy` file is preferable because symbols or relative code base is used instead of an absolute code base. This process has many advantages. Grant the permissions that are defined in the `was.policy` to the specific enterprise application only, while permissions in the `app.policy` file apply to all the enterprise applications that run on the node where the `app.policy` file belongs.

Refer to the *Securing applications and their environment* PDF for more details on policy management.

The following example illustrates the migration of a Java 2 security policy from a previous release. The contents include the Java 2 security policy file for the `app1.ear` enterprise application and the system permissions, which are permissions that are granted to the Java virtual machine (JVM) and the product server runtime.

The default location for the Java 2 security policy file is *profile_root*`/properties/java.policy`. Default permissions are omitted for clarity:

```
// For product Samples
   grant codeBase "file:${app_server_root}/installedApps/app1.ear/-" {
     permission java.security.SecurityPermission "printIdentity";
     permission java.io.FilePermission "${app_server_root}${/}temp${/}somefile.txt",
       "read";
   };
```

For clarity of illustration, all the permissions are migrated as the application level permissions in this example. However, you can grant permissions at a more granular level at the component level (Web, enterprise beans, connector or utility Java archive (JAR) component level) or you can grant permissions to a particular component.

Procedure

1. Ensure that Java 2 security is disabled on the application server.
2. Create a new `was.policy` file, if the file is not present, or update the `was.policy` file for migrated applications in the configuration repository with the following contents:

   ```
   grant codeBase "file:${application}" {
        permission java.security.SecurityPermission "printIdentity";
        permission java.io.FilePermission "
                ${user.install.root}${/}temp${/}somefile.txt", "read";
      };
   ```

   The third and fourth lines in the previous code sample are presented on two lines for illustrative purposes only.

The was.policy file is located in the *profile_root*/config/cells/*cell_name*/applications/app.ear/deployments/app/META-INF/ directory.

3. Use an assembly tool to attach the was.policy file to the enterprise archive (EAR) file.

   You also can use an assembly tool to validate the contents of the was.policy file. For more information, see the *Securing applications and their environment* PDF.

4. Validate that the enterprise application does not require additional permissions to the migrated Java 2 security permissions and the default permissions set declared in the ${user.install.root}/config/cells/*cell_name*/nodes/*node_name*/app.policy file. This validation requires code review, code inspection, application documentation review, and sandbox testing of migrated enterprise applications with Java 2 security enabled in a preproduction environment. Refer to developer kit APIs protected by Java 2 security for information about which APIs are protected by Java 2 security. If you use third-party libraries, consult the vendor documentation for APIs that are protected by Java 2 security. Verify that the application is granted all the required permissions, or it might fail to run when Java 2 security is enabled.

5. Perform preproduction testing of the migrated enterprise application with Java 2 security enabled. Enable trace for the WebSphere Application Server Java 2 security manager in a preproduction testing environment with the following trace string: com.ibm.ws.security.core.SecurityManager=all=enabled. This trace function can be helpful in debugging the AccessControlException exception that is created when an application is not granted the required permission or some system code is not properly marked as privileged. The trace dumps the stack trace and permissions that are granted to the classes on the call stack when the exception is created.

   For more information, see the *Securing applications and their environment* PDF.

   Note: Because the Java 2 security policy is much more stringent compared with previous releases, the administrator or deployer must review their enterprise applications to see if extra permissions are required before enabling Java 2 security. If the enterprise applications are not granted the required permissions, they fail to run.

Migrating with Tivoli Access Manager for authentication enabled

When Tivoli® Access Manager security is configured for your existing environment and security is enabled, you can migrate to WebSphere Application Server, Version 8.5.

Before you begin

Your profiles must be migrated using the migration tools to migrate product configurations.

Important: Do not restart the WebSphere Application Server Version 8.5 server until after performing the following procedure. The migration tools omit some files that enable the server to start correctly.

About this task

After migrating your profiles, additional steps are required when Tivoli Access Manager security is configured.

Note: WebSphere Application Server Version 8.0 and above hosts Tivoli Access Manager specific files under the %WAS_HOME%/tivoli/tam directory. In previous versions, these files were hosted under the %WAS_HOME%/java/jre/ hierarchy.

Note: In the following steps, %WASX% refers to the installation root of the source WebSphere Application Server product, and %WAS8% refers to the installation root of the target WebSphere Application Server product (the Version 8.0 installation root).

Procedure

1. Copy the following files from the source location to target location.

Table 9. Files to copy from the source location to the target location. Files to copy from the source location to the target location

Source Location	Target Location
%WASX%\java\jre\PDPerm.properties	%WAS8%\tivoli\tam\PDPerm.properties
%WASX%\java\jre\lib\security\PdPerm.ks (if found)	%WAS8%\tivoli\tam\lib\security\PdPerm.ks
%WASX%\java\jre\lib\PdPerm.ks (if found)	%WAS8%\tivoli\tam\PdPerm.ks
%WASX%\java\jre\PolicyDirector\PDCA.ks	%WAS8%\tivoli\tam\PolicyDirector\PDCA.ks
%WASX%\java\jre\PolicyDirector\PD.properties	%WAS8%\tivoli\tam\PolicyDirector\PD.properties
%WASX%\java\jre\PolicyDirector\etc\pdjrte_paths	%WAS8%\tivoli\tam\PolicyDirector\etc\pdjrte_paths
%WASX%\java\jre\PolicyDirector\etc\pdjrte_mapping	%WAS8%\tivoli\tam\PolicyDirector\etc\pdjrte_mapping

2. Edit the `PD.properties` file, and change the following configuration settings:

```
appsvr-plcysvrs=null\:0:\:1
config_type=standalone
```

   Make the appropriate changes to point to your Tivoli Access Manager Policy Server, for example:

```
appsvr-plcysvrs=pdmgrd.test.gc.au.ibm.com\:7135\:1
config_type=full
```

3. Edit the following four files on the target system and make sure that all of the path references are corrected:
   - `%WAS8%/tivoli/tam/PdPerm.properties`
   - `%WAS8%/tivoli/tam/PolicyDirector/PD.properties`
   - `%WAS8%/tivoli/tam/PolicyDirector/etc/pdjrte_paths`
   - `%WAS8%/tivoli/tam/PolicyDirector/etc/pdjrte_mapping`

   When you correct the paths, complete the following steps in order:
   a. Ensure that all references from `%WASX%/java/jre/PolicyDirector` are changed to `%WAS8%/tivoli/tam/PolicyDirector`.
   b. Ensure that all references (in the `PdPerm.properties` file) from the`%WASX%/java/jre/[security]/PdPerm.ks` file are changed to `%WAS8%/tivoli/tam/pdPerm.ks`.
   c. Ensure that all remaining references from `%WASX%/java/jre` are changed to `%WAS8%/java/jre`.
   d. Edit the `%WAS8%/tivoli/tam/PolicyDirector/etc/pdjrte_mapping` file. It contains the JRE->JRE mapping: `%WAS8%/java/jre=%WAS8%/java/jre`.

      Change this mapping to JRE->tivoli/tam: `%WAS8%/java/jre=%WAS8%/tivoli/tam`.

What to do next

Also see Migrating with Tivoli Access Manager for authentication enabled on multiple nodes for more information.

Migrating unrestricted jurisdiction policy files, local_policy.jar and US_export_policy.jar

You can migrate the unrestricted jurisdiction policy files, `local_policy.jar` and `US_export_policy.jar`.

About this task

If you want to use encryption keys that are greater than 128-bits, you must use the unrestricted jurisdiction policy files, `local_policy.jar` and `US_export_policy.jar`.

The files are located in the [*WAS_HOME*`/java/jre/lib/security`] directory.

If your back-level version of WebSphere Application Server is using the unrestricted jurisdiction policy files, you must perform special steps to migrate these files to your new version of WebSphere Application Server. If you are not using the unrestricted jurisdiction policy files, you do not need to take any action.

Procedure

1. Before migrating, copy the modified `local_policy.jar` file to a temporary location.
2. Migrate the WebSphere Application Server installation.
3. Copy the modified `local_policy.jar` file from step 1 to the following directory on the new WebSphere Application Server installation: *WAS_HOME*`/java/jre/lib/security` .
4. Start the new WebSphere Application Server installation as normal.

Preparing for security at installation time

Complete the following tasks to implement security before, during, and after installing WebSphere Application Server.

Procedure

1. Secure your environment before installation. This step describes how to perform WebSphere Application Server installation with proper authority on different platforms. For more information refer to "Securing your environment before installation."
2. Prepare the operating system for installation of WebSphere Application Server. This step describes how to prepare the different operating systems for installation of WebSphere Application Server. For more information, see "Preparing the operating system for product installation" n the InfoCenter.
3. Migrate security configurations from previous releases during installation, when you are prompted to do so. This step describes how to migrate security configurations from a previous release of WebSphere Application Server to WebSphere Application Server Version 8.5.

   For more information, see "Migrating product configurations" in the InfoCenter.
4. Optional: You can create a profile during install time. If you elect to do so, administrative security is enabled for that profile "out of the box" by default. A panel is displayed during profile creation time and **enabling administrative security** is selected by default. If you elect to keep this as the default, you must supply an administrative user ID and password. This user ID is created in a federated repository, which is the default user registry when enabling administrative security at profile creation time.
5. If you go into the advanced profile creation, a panel is available for changing the default settings for your certificate, a *root certificate* (used to sign your personal certificate) and a *personal certificate* (used to sign/encrypt data over the network). Ensure that the root certificate has a long lifetime and the personal certificate a shorter one. Import your own personal certificate and or root certificate. If your personal certificate is signed by the certificate authority (CA), it is not important to change your root certificate. You should also change the default keystore password to something more secure.
6. Secure your environment after installation. This step provides information on how to protect password information after you install WebSphere Application Server. For more information, see "Securing your environment after installation" on page 68.

Securing your environment before installation

The following instructions explain how to perform a product installation with proper authority.

About this task

Complete the following steps when you plan to use the local operating system as your user registry:

Procedure

- Windows Secure your environment.

1. The Windows Administrative Tools selection can be found under the Windows Control Panel, depending on which version of Windows you have installed. Under Administrative Tools, select **Local Security Policy** (for domain configuration, select **Domain Security Policies**, instead).
2. From the Local Security Settings panel, click **Local Policies > User Rights Assignment** and add **Login as a service** rights to the login ID. The logon user must be a member of the administrator group with the rights of **Log on as a service**.

- AIX HP-UX Linux Solaris Secure your environment.
  1. Log on as **root** and verify that the umask value is **022**.
  2. To verify that the umask value is **022**, run the **umask** command.
  3. To set up the umask value as **022**, run the **umask 022** command.
  4. Linux Solaris Optional: Make sure that the `/etc` directory contains a shadow password file. The shadow password file is named shadow and is in the `/etc` directory. If the shadow password file does not exist, an error occurs after enabling administrative security and configuring the user registry as local operating system.
  5. To create the shadow file, run the **pwconv** command (without any parameters). This command creates an `/etc/shadow` file from the `/etc/passwd` file. After creating the shadow file, you can configure local operating system security.

Results

These steps help secure your environment when you use the local operating system as your user registry

Securing your environment after installation

WebSphere Application Server depends on several configuration files that are created during installation. These files contain password information and need protection. Although the files are protected to a limited degree during installation, this basic level of protection is probably not sufficient for your site. You should verify that these files are protected in compliance with the policies of your site.

Before you begin

Note: A Kerberos keytab configuration file contains a list of keys that are analogous to user passwords. The default keytab file is krb5.keytab. It is important for hosts to protect their Kerberos keytab files by storing them on the local disk, which makes them readable only by authorized users.

The files in the *app_server_root*`/profiles/`*`profile_name`*`/config` and *app_server_root*`/profiles/`*`profile_name`*`/properties` need protection. For example, give permission to the user who logs onto the system for WebSphere Application Server primary administrative tasks. Other users or groups, such as WebSphere Application Server console users and console groups need permissions as well.

Procedure

- Windows Secure files on a Windows system:
  1. Open the browser for a view of the files and directories on the machine.
  2. Locate and right-click the file or the directory that you want to protect.
  3. Click **Properties**.
  4. Click the **Security** tab.
  5. Remove the `Everyone` entry and any other user or group that you do not want to have access to the file.
  6. Add the users who can access the files with the proper permission.
- AIX HP-UX Solaris Secure files on UNIX systems. This procedure applies only to the ordinary UNIX file system. If your site uses access-control lists, secure the files by using that mechanism. Any site-specific requirements can affect the owner, group, and corresponding privileges; for example, on the AIX® platform.

1. Go to the *install_root* directory and change the ownership of the directory configuration and properties to the user who logs onto the system for WebSphere Application Server primary administrative tasks. Run the following command: `chown -R logon_name directory_name`

   Where:

   – *login_name* is a specified user or group
   – *directory_name* is the name of the directory that contains the files

   It is recommended that you assign ownership of the files that contain password information to the user who runs the application server. If more than one user runs the application server, provide permission to the group in which the users are assigned in the user registry.
2. Set up the permission by running the following command: `chmod -R 770 directory_name`.
3. Go to the *app_server_root*/profiles/*profile_name*/properties directory and set the file permissions. Set the access permissions for the following files as it pertains to your security guidelines:
   – TraceSettings.properties
   – client.policy
   – client_types.xml
   – ipc.client.props
   – sas.client.props
   – sas.stdclient.properties
   – sas.tools.properties
   – soap.client.props
   – wsadmin.properties
   – wsjaas_client.conf

   For example, you might issue the following command: `chmod 770 file_name` where *file_name* is the name of the file listed previously in the install_root/profiles/profile_name/properties directory. These files contain sensitive information such as passwords.

   Note: If you enabled Kerberos authentication or SPNEGO web authentication, set the access permissions for the following files as it pertains to your security guidelines: the Kerberos configuration file (**krb5.conf** or **krb5.ini**) and the Kerberos keytab file.
4. Create a group for WebSphere Application Server and put the users who perform full or partial WebSphere Application Server administrative tasks in that group.
5. If you want to use WebSphere MQ as a Java Messaging Service (JMS) provider, restrict access to the /var/mqm directories and log files used. Give write access to the user ID mqm or members of the mqm user group only.

Results

After securing your environment, only the users with permission can access the files. Failure to adequately secure these files can lead to a breach of security in your WebSphere Application Server applications.

What to do next

If failures occur that are caused by file accessing permissions, check the permission settings.

Enabling security

The following provides information on how to configure security when security was not enabled during the WebSphere Application Sever profile creation.

Before you begin

When you are installing WebSphere Application Server, it is recommended that you install with security enabled. By design, this option ensures that everything has been properly configured. By enabling security, you protect your server from unauthorized users and are then able to provide application isolation and requirements for authenticating application users.

It is helpful to understand security from an infrastructure perspective so that you know the advantages of different authentication mechanisms, user registries, authentication protocols, and so on. Picking the right security components to meet your needs is a part of configuring security. The following sections help you make these decisions.

After you understand the security components, you can proceed to configure security in WebSphere Application Server.

Procedure

1. Start the WebSphere Application Server administrative console.

   If security is currently disabled, you are prompted for a user ID. Log in with any user ID. However, if security is currently enabled, you are prompted for both a user ID and a password. Log in with a predefined administrative user ID and password.
2. Click **Security > Global security**.

   Use the Security Configuration Wizard, or configure security manually. The configuration order is not important.

   gotcha: You must separately enable administrative security, and application security. Because of this split, WebSphere Application Server clients must know whether application security is disabled at the target server. Administrative security is enabled, by default. Application security is disabled, by default. Before you attempt to enable application security on the target server, verify that administrative security is enabled on that server. Application security can be in effect only when administrative security is enabled.

   For more information on manual configuration, see Authenticating users.
3. Configure the user account repository. For more information, see “Selecting a registry or repository” on page 159. On the Global security panel, you can configure user account repositories such as federated repositories, local operating system, stand-alone Lightweight Directory Access Protocol (LDAP) registry, and stand-alone custom registry.

   Note: You can choose to specify either a server ID and password for interoperability or enable a WebSphere Application Server installation to automatically generate an internal server ID. For more information about automatically generating server IDs, see “Local operating system settings” on page 169.

   One of the details common to all user registries or repositories is the *Primary administrative user name*. This ID is a member of the chosen repository, but also has special privileges in WebSphere Application Server. The privileges for this ID and the privileges that are associated with the administrative role ID are the same. The Primary administrative user name can access all of the protected administrative methods.

   Windows The ID must not be the same name as the machine name of your system because the repository sometimes returns machine-specific information when querying a user of the same name.

   In stand-alone LDAP registries, verify that the Primary administrative user name is a member of the repository and not just the LDAP administrative role ID. The entry must be searchable.

   The Primary administrative user name does **not** run WebSphere Application Server processes. Rather, the process ID runs the WebSphere Application Server processes.

   The *process ID* is determined by the way the process starts. For example, if you use a command line to start processes, the user ID that is logged into the system is the process ID. If running as a

service, the user ID that is logged into the system is the user ID running the service. If you choose the local operating system registry, the process ID requires special privileges to call the operating system APIs. The process ID must have the following platform-specific privileges:

- Windows **Act as Part of Operating System** privileges
- AIX HP-UX Solaris **Root** privileges

4. Select the **Set as current** option after you configure the user account repository. When you click **Apply** and the Enable administrative security option is set, a verification occurs to see if an administrative user ID has been configured and is present in the active user registry. The administrative user ID can be specified at the active user registry panel or from the console users link. If you do not configure an administrative ID for the active user registry, the validation fails.

   Note: When you switch user registries, the `admin-authz.xml` file should be cleared of existing administrative ids and application names. Exceptions will occur in the logs for ids that exist in the `admin-authz.xml` file but do not exist in the current user registry.

5. Configure the authentication mechanism.

   Configure Lightweight Third-Party Authentication (LTPA) or Kerberos, which is new to this release of WebSphere Application Server, under Authentication mechanisms and expiration. LTPA credentials can be forwarded to other machines. For security reasons, credential expire; however, you can configure the expiration dates on the console. LTPA credentials enable browsers to visit different product servers, which means you do not have to authenticate multiple times. For more information, see Configuring the Lightweight Third Party Authentication mechanism

   Note: You can configure Simple WebSphere Authentication Mechanism (SWAM) as your authentication mechanism. However, SWAM was deprecated in WebSphere Application Server Version 8.5 and will be removed in a future release. SWAM credentials are not forwardable to other machines and for that reason do not expire.

6. Optional: Import and export the LTPA keys for cross-cell single Sign-on (SSO) between cells. For more information, see the following articles:
   - Exporting Lightweight Third Party Authentication keys.
   - Importing Lightweight Third Party Authentication keys

   gotcha: If one of the cells you are connecting to resides on a Version 6.0.x system, see the topic Configuring Lightweight Third Party Authentication keys in the Version 6.0.x Information Center for more information.

7. Configure the authentication protocol for special security requirements from Java clients, if needed.

   You can configure Common Secure Interoperability Version 2 (CSIv2) through links on the Global security panel. The Security Authentication Service (SAS) protocol is provided for backwards compatibility with previous product releases, but is deprecated. Links to the SAS protocol panels display on the Global security panel if your environment contains servers that use previous versions of WebSphere Application Server and support the SAS protocol. For details on configuring CSIv2 or SAS, see the article, “Configuring Common Secure Interoperability Version 2 (CSIV2) inbound and outbound communication settings” on page 486.

   Attention: IBM no longer ships or supports the Secure Authentication Service (SAS) IIOP security protocol. It is recommended that you use the Common Secure Interoperability version 2 (CSIv2) protocol.

8. Secure Socket Layers (SSL) is pre-configured by default, changes are not necessary unless you have custom SSL requirements. You can modify or a create a new SSL configuration. This action protects the integrity of the messages sent across the Internet. The product provides a centralized location to configure SSL configurations that the various WebSphere Application Server features that use SSL can utilize, including the LDAP registry, web container and the RMI/IIOP authentication protocol (CSIv2). For more information, see “Creating a Secure Sockets Layer configuration” on page 713. After you modify a configuration or create a new configuration, specify it on the **SSL configurations** panel. To get to the SSL configurations panel, complete the following steps:

a. Click **Security > SSL certificate and key management**.
b. Under Configuration settings, click **Manage endpoint security configurations >** *configuration_name*.
c. Under Related items for each scope (for example, node, cluster, server), select one of the many configuration links that can be scoped to the resource you are visiting.

You can either edit the DefaultSSLConfig file or create a new SSL configuration with a new alias name. If you create a new alias name for your new keystore and truststore files, change every location that references the DefaultSSLConfig SSL configuration alias. The following list specifies the locations of where the SSL configuration repertoire aliases are used in the WebSphere Application Server configuration.

For any transports that use the new network input/output channel chains, including HTTP and Java Message Service (JMS), you can modify the SSL configuration repertoire aliases in the following locations for each server:

- Click **Server > Application server >** *server_name*. Under Communications, click **Ports**. Locate a transport chain where SSL is enabled and click **View associated transports**. Click *transport_channel_name*. Under Transport Channels, click **SSL Inbound Channel (SSL_2)**.

For the Object Request Broker (ORB) SSL transports, you can modify the SSL configuration repertoire aliases in the following locations. These configurations are for the server-level for WebSphere Application Server and WebSphere Application Server, Express and the cell level for WebSphere Application Server, Network Deployment.

- Click **Security > Global security**. Under RMI/IIOP security, click **CSIv2 inbound communications**.
- Click **Security > Global security**. Under RMI/IIOP security, click **CSIv2 outbound communications**.

For the Lightweight Directory Access Protocol (LDAP) SSL transport, you can modify the SSL configuration repertoire aliases by clicking **Security > Global security**. Under User account repository, click the **Available realm definitions** drop-down list, and select **Standalone LDAP registry**.

9. Click **Security > Global security** to configure the rest of the security settings and enable security. For information about these settings, see “Global security settings” on page 86.
10. Validate the completed security configuration by clicking **OK** or **Apply**. If problems occur, they display at the top of the console page in red type.
11. If there are no validation problems, click **Save** to save the settings to a file that the server uses when it restarts. Saving writes the settings to the configuration repository.

    Important: If you do not click **Apply** or **OK** in the Global security panel before you click **Save**, your changes are not written to the repository. The server must be restarted for any changes to take effect when you start the administrative console.
12. Start the WebSphere Application Server administrative console.

    If security is currently disabled, log in with any user ID. If security is currently enabled, log in with a predefined administrative ID and password. This ID is typically the server user ID that is specified when you configured the user registry.

Administrative security

Administrative security determines whether security is used at all, the type of registry against which authentication takes place, and other values, many of which act as defaults. Proper planning is required because incorrectly enabling administrative security can lock you out of the administrative console or cause the server to end abnormally.

Note: It is strongly recommended that you allow the default installation to install administrative security as on by default.

Administrative security can be thought of as a "big switch" that activates a wide variety of security settings for WebSphere Application Server. Values for these settings can be specified, but they will not take effect until administrative security is activated. The settings include the authentication of users, the use of Secure Sockets Layer (SSL), and the choice of user account repository. In particular, application security, including authentication and role-based authorization, is not enforced unless administrative security is active. Administrative security is enabled by default.

Note: Administrative security need not be activated in order for WebSphere applications to make use of JSSE methods to encrypt communication to remote sites.

Administrative security represents the security configuration that is effective for the entire security domain. A *security domain* consists of all of the servers that are configured with the same user registry *realm* name. In some cases, the realm can be the machine name of a local operating system registry. In this case, all of the application servers must reside on the same physical machine. In other cases, the realm can be the machine name of a stand-alone Lightweight Directory Access Protocol (LDAP) registry.

The basic requirement for a security domain is that the access ID that is returned by the registry or repository from one server within the security domain is the same access ID as that returned from the registry or repository on any other server within the same security domain. The *access ID* is the unique identification of a user and is used during authorization to determine if access is permitted to the resource.

The administrative security configuration applies to every server within the security domain.

Why turn on administrative security?

Turning on administrative security activates the settings that protect your server from unauthorized users. Administrative security is enabled by default during the profile creation time. There might be some environments where no security is needed such as a development system. On these systems you can elect to disable administrative security. However, in most environments you should keep unauthorized users from accessing the administrative console and your business applications. Administrative security must be enabled to restrict access.

What does administrative security protect?

The configuration of administrative security for a security domain involves configuring the following technologies:

- Authentication of HTTP clients
- Authentication of IIOP clients
- Administrative console security
- Naming security
- Use of SSL transports
- Role-based authorization checks of servlets, enterprise beans, and mbeans
- Propagation of identities (RunAs)
- The common user registry
- The authentication mechanism
- Other security information that defines the behavior of a security domain includes:
  - The authentication protocol (Remote Method Invocation over the Internet Inter-ORB Protocol (RMI/IIOP) security)
  - Other miscellaneous attributes

Note: It is recommended that before registering a node with an administrative agent process, that you first have administrative security enabled in the administrative agent and base profile. Once you register a profile with the administrative agent, the state of administrative security enablement cannot be changed.

Application security

Application security enables security for the applications in your environment. This type of security provides application isolation and requirements for authenticating application users

In previous releases of WebSphere Application Server, when a user enabled global security, both administrative and application security were enabled. In WebSphere Application Server Version 6.1, the previous notion of global security is split into administrative security and application security, each of which you can enable separately.

As a result of this split, WebSphere Application Server clients must know whether application security is disabled at the target server. Administrative security is enabled, by default. Application security is disabled, by default. Before you can enable application security, you must verify that administrative security is enabled. Application security is in effect only when administrative security is enabled.

An Application Server Enablement Tag, which is specific to WebSphere Application Server, is imported into the Interoperable Object Reference (IOR) to indicate if application security is disabled for the server where the object lives. This tag is server-specific and enables clients to know when application security is disabled at the target server of its request.

For web resources, when application security is enabled, security constraints on those resources in web.xml are enforced. When accessing a protected resource, a web client is prompted for authentication.

For enterprise bean resources, when application security is disabled, the client Common Secure Interoperability version 2 (CSIv2) code ignores the CSIv2 security tags for objects that are unknown system objects. When pure clients see that application security is disabled, these clients prompt for naming lookups, but do not prompt for enterprise bean operations.

Java 2 security

Java 2 security provides a policy-based, fine-grain access control mechanism that increases overall system integrity by checking for permissions before allowing access to certain protected system resources. Java 2 security guards access to system resources such as file I/O, sockets, and properties. Java 2 Platform, Enterprise Edition (J2EE) security guards access to web resources such as servlets, JavaServer Pages (JSP) files and Enterprise JavaBeans (EJB) methods.

Because Java 2 security is relatively new, many existing or even new applications might not be prepared for the very fine-grain access control programming model that Java 2 security is capable of enforcing. Administrators need to understand the possible consequences of enabling Java 2 security if applications are not prepared for Java 2 security. Java 2 security places some new requirements on application developers and administrators.

Note: The application server does not support a custom Java security manager implementation.

Java 2 security for deployers and administrators

Although Java 2 security is supported, it is disabled by default. You can configure Java 2 security and administrative security independently of one another. Disabling administrative security does not disable Java 2 security automatically. You need to explicitly disable it.

If your applications, or third-party libraries are not ready, having Java 2 security enabled causes problems. You can identify these problems as Java 2 security AccessControlExceptions in the system log or trace

files. If you are unsure about the Java 2 security readiness of your applications, disable Java 2 security initially to get your application installed and verify that it is working properly.

The policy embodied by these policy files cannot be made more restrictive because the product might not have the necessary Java 2 security doPrivileged APIs in place. The restrictive policy is the default policy. You can grant additional permissions, but you cannot make the default more restrictive because AccessControlExceptions exceptions are generated from within WebSphere Application Server. The product does not support a more restrictive policy than the default that is defined in the policy files previously mentioned.

Several policy files are used to define the security policy for the Java process. These policy files are static (code base is defined in the policy file) and in the default policy format provided by the IBM Developer Kit, Java Technology Edition. For enterprise application resources and utility libraries, WebSphere Application Server provides dynamic policy support. The code base is dynamically calculated based on deployment information and permissions are granted based on template policy files during runtime. Refer to the “Java 2 security policy files” on page 79 for more information.

Syntax errors in the policy files cause the application server process to fail, so edit these policy files carefully.

If an application is not prepared for Java 2 security, if the application provider does not provide a `was.policy` file as part of the application, or if the application provider does not communicate the expected permissions the application is likely to cause Java 2 security access control exceptions at runtime. It might not be obvious that an application is not prepared for Java 2 security. Several run-time debugging aids help troubleshoot applications that might have access control exceptions. See the Java 2 security debugging aids for more details. See “Handling applications that are not Java 2 security ready” on page 77 for information and strategies for dealing with such applications.

It is important to note when Java Security is enabled in the administrative security settings, the installed security manager does not currently check modifyThread and modifyThreadGroup permissions for non-system threads. Allowing web and Enterprise JavaBeans (EJB) application code to create or modify a thread can have a negative impact on other components of the container and can affect the capability of the container to manage enterprise bean life cycles and transactions.

Java 2 security for application developers

Application developers must understand the permissions that are granted in the default WebSphere policy and the permission requirements of the SDK APIs that their application calls to know whether additional permissions are required. The Permissions in the Java 2 SDK reference in the resources section describes which APIs require which permission.

Application providers can assume that applications have the permissions granted in the default policy previously mentioned. Applications that access resources not covered by the default WebSphere policy are required to grant the additional Java 2 security permissions to the application.

While it is possible to grant the application additional permissions in one of the other dynamic WebSphere policy files or in one of the more traditional `java.policy` static policy files, the `was.policy` file, which is embedded in the EAR file ensures the additional permissions are scoped to the exact application that requires them. Scoping the permission beyond the application code that requires it can permit code that normally does not have permission to access particular resources.

If an application component is being developed, like a library that might actually be included in more than one `.ear` file, then the library developer needs to document the required Java 2 permissions that are required by the application assembler. There is no `was.policy` file for library-type components. The developer must communicate the required permissions through application programming interface (API) documentation or some other external documentation.

If the component library is shared by multiple enterprise applications, the permissions can be granted to all enterprise applications on the node in the `app.policy` file.

Note: Updates to the `app.policy` file only apply to the enterprise applications on the node to which the `app.policy` file belongs.

If the permission is only used internally by the component library and the application is never granted access to resources that are protected by the permission, it might be necessary to mark the code as `privileged`. Refer to the, AccessControlException, topic for more details. However, improperly inserting a doPrivileged call might open up security holes. Understand the implication of doPrivileged call to make a correct judgement.

The section on Dynamic policy files in "Java 2 security policy files" on page 79 describes how the permissions in the `was.policy` files are granted at runtime.

Developing an application to use with Java 2 security might be a new skill and impose a security awareness not previously required of application developers. Describing the Java 2 security model and the implications on application development is beyond the scope of this section. The following URL can help you get started: http://java.sun.com/j2se/1.5.0/docs/guide/security/index.html.

Debugging Aids

The WebSphere Application Server SYSOUT file and the com.ibm.websphere.java2secman.norethrow property are the two primary aids for debugging.

The WebSphere System Log or Trace Files

The AccessControl exception that is logged in the system log or trace files contains the permission violation that causes the exception, the exception call stack, and the permissions granted to each stack frame. This information is usually enough to determine the missing permission and the code requiring the permission.

The com.ibm.websphere.java2secman.norethrow property

When Java 2 security is enabled in WebSphere Application Server, the security manager component creates a java.security.AccessControl exception when a permission violation occurs. This exception, if not handled, often causes a run-time failure. This exception is also logged in the SYSOUT file.

However, when the Java virtual machine com.ibm.websphere.java2secman.norethrow property is set and has a value of `true`, the security manager does not create the AccessControl exception. This information is logged.

This property is intended for a sandbox or debug environment because it instructs the security manager not to create the AccessControl exception. Java 2 security is not enforced. Do not use this property in a production environment where a relaxed Java 2 security environment weakens the integrity that Java 2 security is intended to produce.

This property is valuable in a sandbox or test environment where the application can be thoroughly tested and where the system log or trace files can be inspected for AccessControl exceptions. Because this property does not create the AccessControl exception, it does not propagate the call stack and does not cause a failure. Without this property, you have to find and fix AccessControl exceptions one at a time.

Handling applications that are not Java 2 security ready

If the increased system integrity that Java 2 security provides is important, then contact the application provider to have the application support Java 2 security or at least communicate the required additional permissions beyond the default WebSphere Application Server policy that must be granted.

The easiest way to deal with such applications is to disable Java 2 security in WebSphere Application Server. The downside is that this solution applies to the entire system and the integrity of the system is not as strong as it might be. Disabling Java 2 security might not be acceptable depending on the organization security policies or risk tolerances.

Another approach is to leave Java 2 security enabled, but to grant either just enough additional permissions or grant all permissions to just the problematic application. Granting permissions however, might not be a trivial thing to do. If the application provider has not communicated the required permissions in some way, no easy way exists to determine what the required permissions are and granting all permissions might be the only choice. You minimize this risk by locating this application on a different node, which might help isolate it from certain resources. Grant the java.security.AllPermission permission in the `was.policy` file that is embedded in the application `.ear` file, for example:

```
grant codeBase "file:${application}" {
          permission java.security.AllPermission;
       };
```

The server.policy file

The `server.policy` file is located in the *app_server_root*/`properties/` directory.

This policy defines the policy for the WebSphere Application Server classes. At present, all the server processes on the same installation share the same `server.policy` file. However, you can configure this file so that each server process can have a separate `server.policy` file. Define the policy file as the value of the java.security.policy Java system properties . For details of how to define Java system properties, refer to the Process definition section of the Manage application servers file.

The `server.policy` file is not a configuration file managed by the repository and the file replication service. Changes to this file are local and do not get replicated to other machines. Use the `server.policy` file to define Java 2 security policy for server resources. Use the `app.policy` file (per node) or the `was.policy` file (per enterprise application) to define Java 2 security policy for enterprise application resources.

Note: Updates to the `app.policy` file only apply to the enterprise applications on the node to which the app.policy file belongs.

The java.policy file

The file represents the default permissions that are granted to all classes. The policy of this file applies to all the processes launched by the Java Virtual Machine in the WebSphere Application Server.

The `java.policy` file is located in the *app_server_root*/`java/lib/security` directory.

Troubleshooting

Error message CWSCJ0314E

Symptom:

`Error message CWSCJ0314E`: Current® Java 2 security policy reported a potential violation of Java 2 security permission. Refer to Problem Determination Guide for further information.{0}Permission\ :{1}Code\:{2}{3}Stack Trace\:{4}Code Base Location\:{5} Current Java 2 security policy reported a potential violation of Java 2 Security Permission. Refer to Problem Determination Guide for further information.{0}Permission\:{1}Code\:{2}{3}Stack Trace\:{4}Code Base Location\:{5}

Problem:

The Java security manager checkPermission method reported a security exception on the subject permission with debugging information. The reported information can be different with respect to the system configuration. This report is enabled by either configuring a Reliability Availability Service Ability (RAS) trace into debug mode or specifying a Java property.

See Enabling trace for information on how to configure RAS trace in debug mode.

Specify the following property in the JVM Settings panel from the administrative console: `java.security.debug`. Valid values include:

access
Print all debug information including: required permission, code, stack, and code base location.

stack Print debug information including: required permission, code, and stack.

failure Print debug information including: required permission and code.

Recommended response:

The reported exception might be critical to the secure system. Turn on security trace to determine the potential code that might have violated the security policy. After the violating code is determined, verify if the attempted operation is permitted with respect to Java 2 security, by examining all applicable Java 2 security policy files and the application code.

If the application is running with Java Mail, this message might be benign. You can update the `was.policy` file to grant the following permissions to the application:

```
permission  java.io.FilePermission  "${user.home}${/}.mailcap",  "read";
permission  java.io.FilePermission  "${user.home}${/}.mime.types",  "read";
permission  java.io.FilePermission  "${java.home}${/}lib${/}mailcap",  "read";
permission  java.io.FilePermission  "${java.home}${/}lib${/}mime.types",  "read";
```

SecurityException - Access denied

Symptom:

If Java security is enabled, and permissions to read the jaxm.properties file is not granted, when a SOAPFactory instance is created through a call to javax.xml.soap.SOAPFactory.newInstance(), or a MessageFactory instance is created through a call to MessageFactory.newInstance(), a SecurityException exception occurs, and the following exception is written to the system log:

```
Permission:

      /opt/IBM/WebSphere/AppServer/java/jre/lib/jaxm.properties : access denied
(java.io.FilePermission /opt/IBM/WebSphere/AppServer/java/jre/lib/jaxm.properties
read)

Code:

     com.ibm.ws.wsfvt.test.binding.addr1.binder.AddressBinder
in  {file:/opt/IBM/WebSphere/AppServer/profiles/AppSrv01/installedApps/
ahp6405Node01Cell/DataBinding.ear/address1.war/WEB-INF/lib
/addressbinder1.jar}

Stack Trace:

java.security.AccessControlException: access denied (java.io.FilePermission
/opt/IBM/WebSphere/AppServer/java/jre/lib/jaxm.properties read)
.
```

Problem:

The Java 2 Security policy reports a potential violation of Java 2 Security permission.

Recommended response:

The SOAPFactory ignores the exception, and continues on to the next means of determining which implementation to load. Therefore, you can ignore the log entry for this security exception.

Because this product uses the SOAPFactory to support other web services technologies, such as WS-Addressing (WS-A), WS-Atomic Transaction (WS-AT), and WS-Notification, you can ignore this SecurityException in any web services application where Java security is enabled.

Messages

Message: CWSCJ0313E: Java 2 security manager debug message flags are initialized\: TrDebug: {0}, Access: {1}, Stack: {2}, Failure: {3}

Problem: Configured values of the valid debug message flags for security manager.

Message: CWSCJ0307E: Unexpected exception is caught when trying to determine the code base location. Exception: {0}

Problem: An unexpected exception is caught when the code base location is determined.

Java 2 security policy files

The Java 2 Platform, Enterprise Edition (J2EE) Version 1.3 and later specifications have a well-defined programming model of responsibilities between the container providers and the application code. Using Java 2 security manager to help enforce this programming model is recommended. Certain operations are not supported in the application code because such operations interfere with the behavior and operation of the containers. The Java 2 security manager is used in the product to enforce responsibilities of the container and the application code.

Note: The application server does not support a custom Java security manager implementation.

This product provides support for policy file management. A number of policy files in the product are either static or dynamic. *Dynamic policy* is a template of permissions for a particular type of resource. No relative code base is defined in the dynamic policy template. The code base is dynamically calculated from the deployment and run-time data.

Static policy files

Table 10. Static policy files.

This table lists the location of the static policy files.

Policy file	Location
java.policy	*app_server_root*/java/jre/lib/security/java.policy. Default permissions are granted to all classes. The policy of this file applies to all the processes launched by WebSphere Application Server.
server.policy	*profile_root*/properties/server.policy. Default permissions are granted to all the product servers.
client.policy	*profile_root*/properties/client.policy. Default permissions are granted for all of the product client containers and applets on a node.

The static policy files are not managed by configuration and file replication services. Changes made in these files are local and are not replicated to other nodes in the WebSphere Application Server, Network Deployment cell.

Dynamic policy files

Table 11. Dynamic policy files.

This table lists the location of the dynamic policy files.

Policy file	Location
spi.policy	*profile_root*/config/cells/*cell_name* /nodes/*node_name*/spi.policy This template is for the Service Provider Interface (SPI) or the third-party resources that are embedded in the product. Examples of SPI are the Java Message Service (JMS) in MQ Series and Java database connectivity (JDBC) drivers. The code base for the embedded resources are dynamically determined from the configuration (resources.xml file) and run-time data, and permissions that are defined in the spi.policy files are automatically applied to these resources and JAR files that are specified in the class path of a resource adapter. The default permission of the spi.policy file is java.security.AllPermissions.
library.policy	*profile_root*/config/cells/*cell_name*/nodes /*node_name*/library.policy This template is for the library (Java library classes). You can define a shared library to use in multiple product applications. The default permission of the library.policy file is empty.
app.policy	*profile_root*/config/cells/*cell_name* /nodes/*node_name*/app.policy The app.policy file defines the default permissions that are granted to all of the enterprise applications running on *node_name* in *cell_name*. **Note:** Updates to the app.policy file only apply to the enterprise applications on the node to which the app.policy file belongs.
was.policy	*profile_root*/config/cells/*cell_name* /applications/*ear_file_name*/deployments/ *application_name*/META-INF/was.policy This template is for application-specific permissions. The was.policy file is embedded in the enterprise archive (EAR) file.
ra.xml	*rar_file_name*/META-INF/was.policy.RAR. This file can have a permission specification that is defined in the ra.xml file. The ra.xml file is embedded in the RAR file.

Grant entries that are specified in the app.policy and was.policy files must have a code base defined. If grant entries are specified without a code base, the policy files are not loaded properly and the application can fail. If the intent is to grant the permissions to all applications, use *file:${application}* as a code base in the grant entry.

Syntax of the policy file

A policy file contains several policy entries. The following example depicts each policy entry format:

```
grant [codebase <Codebase>] {
permission <Permission>;
 permission <Permission>;
permission <Permission>;
};

<CodeBase>:  A URL.
   For example,  "file:${java.home}/lib/tools.jar"
      When [codebase <Codebase>] is not specified, listed
               permissions are applied to everything.
      If URL ends with a JAR file name,  only the classes in the
               JAR file belong to the codebase.
               If URL ends with "/", only the class files in the specified
               directory belong to the codebase.
               If URL ends with "*", all JAR and class files in the specified
               directory belong to the codebase.
               If URL ends with "-", all JAR and class files in the specified
               directory and its subdirectories belong to the codebase.
<Permissions>: Consists from
       Permission Type   : class name of the permission
          Target Name       : name specifying the target
          Actions           : actions allowed on target
   For example,
         java.io.FilePermission "/tmp/xxx", "read,write"
```

Refer to developer kit specifications for the details of each permission.

Syntax of dynamic policy

You can define permissions for specific types of resources in dynamic policy files for an enterprise application. This action is achieved by using *product-reserved symbols*. The reserved symbol scope depends on where it is defined. If you define the permissions in the `app.policy` file, the symbol applies to all the resources on all of the enterprise applications that run on *node_name*. If you define the permissions in the `META-INF/was.policy` file, the symbol applies only to the specific enterprise application. Valid symbols for the code base are listed in the following table:

Table 12. Dynamic policy syntax.

This table describes valid symbols for the code base for dynamic policy files.

Symbol	Meaning
file:${application}	Permissions apply to all the resources within the application
file:${jars}	Permissions apply to all the utility Java archive (JAR) files within the application
file:${ejbComponent}	Permissions apply to the Enterprise JavaBeans (EJB) resources within the application
file:${webComponent}	Permissions apply to the web resources within the application
file:${connectorComponent}	Permissions apply to the connector resources within the application

You can specify the module name for a granular setting, except for these entries that are specified by the code base symbols. For example:

```
grant codeBase "file:DefaultWebApplication.war" {
   permission java.security.SecurityPermission "printIdentity";
 };

grant codeBase "file:IncCMP11.jar" {
permission java.io.FilePermission
"${user.install.root}${/}bin${/}DefaultDB${/}-",
"read,write,delete";
};
```

The sixth and seventh lines in the previous code sample are one continuous line. You can use a relative code base only in the `META-INF/was.policy` file. Several product-reserved symbols are defined to associate the permission lists to a specific type of resources.

Table 13. Dynamic policy syntax.

This table describes several product-reserved symbols that are defined to associate the permission lists to a specific type of resource.

Symbol	Meaning
file:${application}	Permissions apply to all the resources within the application
file:${jars}	Permissions apply to all the utility JAR files within the application
file:${ejbComponent}	Permissions apply to the enterprise beans resources within the application
file:${webComponent}	Permissions apply to the web resources within the application
file:${connectorComponent}	Permissions apply to the connector resources both within the application and in the standalone connector resources.

Five embedded symbols are provided to specify the path and the name for the java.io.FilePermission permission. These symbols enable flexible permission specification. The absolute file path is fixed after the installation of the application.

Table 14. Dynamic policy syntax.

This table describes the embedded symbols that are provided to specify the path and name for the java.io.FilePermission permission.

Symbol	Meaning
${app.installed.path}	Path where the application is installed

Table 14. Dynamic policy syntax (continued).

This table describes the embedded symbols that are provided to specify the path and name for the java.io.FilePermission permission.

Symbol	Meaning
${was.module.path}	Path where the module is installed
${current.cell.name}	Current cell name
${current.node.name}	Current node name
${current.server.name}	Current server name

Attention: Do not use the `${was.module.path}` in the `${application}` entry.

Carefully determine where to add a new permission. An incorrectly specified permission causes an AccessControlException exception. Because dynamic policy resolves the code base at runtime, determining which policy file has a problem is difficult. Add a permission only to the necessary resources. For example, `use ${ejbcomponent}`, and `etc` instead of `${application}`, and update the `was.policy` file instead of the `app.policy` file, if possible.

Static policy filtering

Limited static policy filtering support exists. If the `app.policy` file and the `was.policy` file have permissions that are defined in the `filter.policy` file with thefilterMask keyword, the runtime removes the permissions from the applications and an audit message is logged. However, if the permissions that are defined in the `app.policy` and the `was.policy` files are compound permissions, for example, java.security.AllPermission, the permission is not removed, but a warning message is written to the log file. The policy filtering only supports Developer Kit permissions; the permissions package name begins with `java` or `javax`.

Run-time policy filtering support is provided to force stricter filtering. If the `app.policy` file and the `was.policy` file have permissions that are defined in the `filter.policy` file with the runtimeFilterMask keyword, the runtime removes the permissions from the applications no matter what permissions are granted to the application. For example, even if a `was.policy` file has the java.security.AllPermission permission granted to one of its modules, specified permissions such as the runtimeFilterMask permission are removed from the granted permission during runtime.

Policy file editing

Using the policy tool that is provided by the Developer Kit (*app_server_root*`/java/jre/bin/policytool`), to edit the previous policy files is recommended. For WebSphere Application Server, Network Deployment, extract the policy files from the repository before editing. After the policy file is extracted, use the policy tool to edit the file. Check the modified policy files into the repository and synchronize them with other nodes.

Troubleshooting

To debug the dynamic policy, choose one of three ways to generate the detail report of the AccessControlException exception.

- **Trace** (Configured by RAS trace). Enables traces with the trace specification:

  Attention: The following command is one continuous line

```
com.ibm.ws.security.policy.*=all=enabled:
com.ibm.ws.security.core.SecurityManager=all=enabled
```

- **Trace** (Configured by property). Specifies a Java java.security.debug property. Valid values for the java.security.debug property are as follows:
  - Access. Print all debug information including required permission, code, stack, and code base location.
  - Stack. Print debug information including, required permission, code, and stack.

  – Failure. Print debug information including required permission and code.
- **ffdc**. Enable `ffdc`, modify the `ffdcRun.properties` file by changing `Level=4` and `LAE=true`. Look for an Access Violation keyword in the log file.

Access control exception for Java 2 security

The Java 2 security behavior is specified by its security policy. The security policy is an access-control matrix that specifies which system resources certain code bases can access and who must sign them. The Java 2 security policy is declarative and it is enforced by the java.security.AccessController.checkPermission method.

The following example depicts the algorithm for the java.security.AccessController.checkPermission method. For the complete algorithm, refer to the Java 2 security check permission algorithm in the Security: Resources for learning article.

```
i = m;
while (i > 0) {
 if (caller i's domain does not have the permission)
  throw AccessControlException;
 else if (caller i is marked as privileged)
  return;
 i = i - 1;
};
```

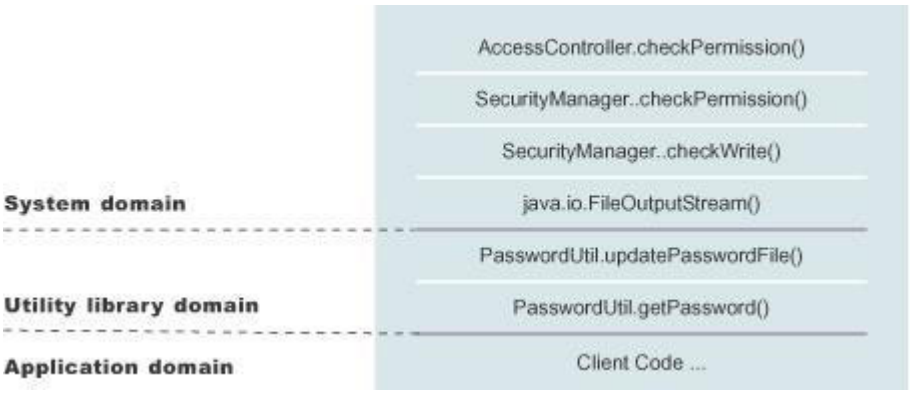
The algorithm requires that all the classes or callers on the call stack have the permissions when a java.security.AccessController.checkPermission method is performed or the request is denied and a java.security.AccessControlException exception is created. However, if the caller is marked as privileged and the class (caller) is granted these permissions, the algorithm returns and does not traverse the entire call stack. Subsequent classes (callers) do not need the required permission granted.

A java.security.AccessControlException exception is created when certain classes on the call stack are missing the required permissions during a java.security.AccessController.checkPermission method. Two possible resolutions to the java.security.AccessControlException exception are as follows:

- If the application is calling a Java 2 security-protected application programming interface (API), grant the required permission to the application Java 2 security policy. If the application is not calling a Java 2 security-protected API directly, the required permission results from the side-effect of the third-party APIs accessing Java 2 security-protected resources.
- If the application is granted the required permission, it gains more access than it needs. In this case, it is likely that the third party code that accesses the Java 2 security-protected resource is not properly marked as privileged.

Example call stack

This example of a call stack indicates where application code is using a third-party API utility library to update the password. The following example is presented to illustrate the point. The decision of where to mark the code as privileged is application-specific and is unique in every situation. This decision requires great depth of domain knowledge and security expertise to make the correct judgement. A number of well written publications and books are available on this topic. Referencing these materials for more detailed information is recommended.



You can use the PasswordUtil utility to change the password of a user. The utility types in the old password and the new password twice to ensure that the correct password is entered. If the old password matches the one stored in the password file, the new password is stored and the password file updates. Assume that none of the stack frame is marked as privileged. According to the java.security.AccessController.checkPermission algorithm, the application fails unless all the classes on the call stack are granted write permission to the password file. The client application does not have permission to write to the password file directly and to update the password file at will.

However, if the PasswordUtil.updatePasswordFile method marks the code that accesses the password file as privileged, then the check permission algorithm does not check for the required permission from classes that call thePasswordUtil.updatePasswordFile method for the required permission as long as the PasswordUtil class is granted the permission. The client application can successfully update a password without granting the permission to write to the password file.

The ability to mark code privileged is very flexible and powerful. If this ability is used incorrectly, the overall security of the system can be compromised and security holes can be exposed. Use the ability to mark code privileged carefully.

Resolution to the java.security.AccessControlException exception

As described previously, you have two approaches to resolve a java.security.AccessControlException exception. Judge these exceptions individually to decide which of the following resolutions is best:
1. Grant the missing permission to the application.
2. Mark some code as privileged, after considering the issues and risks.

Enabling security for the realm

Use this topic to enable IBM WebSphere Application Server security. You must enable administrative security for all other security settings to function.

About this task

WebSphere Application Server uses cryptography to protect sensitive data and to ensure confidentiality and integrity of communications between WebSphere Application Server and other components in the network. Cryptography is also used by Web Services Security when certain security constraints are configured for the web services application.

WebSphere Application Server uses Java Secure Sockets Extension (JSSE) and Java Cryptography Extension (JCE) libraries in the Software Development Kit (SDK) to perform this cryptography. The SDK provides strong but limited jurisdiction policy files. Unrestricted policy files provide the ability to perform full strength cryptography and to improve performance.

Attention: Fix packs that include updates to the Software Development Kit (SDK) might overwrite unrestricted policy files. Back up unrestricted policy files before you apply a fix pack and reapply these files after the fix pack is applied.

WebSphere Application Server provides a SDK 6 that contains strong, but limited jurisdiction policy files. You can download the unrestricted policy files from the following website: IBM developer kit: Security information.

Note: Fix packs that include updates to the Software Development Kit (SDK) might overwrite unrestricted policy files. Back up unrestricted policy files before you apply a fix pack and reapply these files after the fix pack is applied.

Important: Your country of origin might have restrictions on the import, possession, use, or re-export to another country, of encryption software. Before downloading or using the unrestricted policy files, you must check the laws of your country, its regulations, and its policies concerning the import, possession, use, and re-export of encryption software, to determine if it is permitted.

Complete the following steps to download and install the new policy files:

1. Click **Java SE 6**
2. Scroll down the page then click **IBM SDK Policy files**.

   The Unrestricted JCE Policy files for SDK 6 website displays.
3. Click **Sign in** and provide your IBM.com ID and password.
4. Select **Unrestricted JCE Policy files for SDK 6** and click **Continue**.
5. View the license and click **I Agree** to continue.
6. Click **Download Now**.
7. Extract the unlimited jurisdiction policy files that are packaged in the compressed file. The compressed file contains a `US_export_policy.jar` file and a `local_policy.jar` file.
8. In your WebSphere Application Server installation, go to the `$JAVA_HOME/jre/lib/security` directory and back up your `US_export_policy.jar` and `local_policy.jar` files.
9. Replace your `US_export_policy.jar` and `local_policy.jar` files with the two files that you downloaded from the IBM.com website.

Complete the following steps to enable security for the realm:

Procedure

1. Enable security in the WebSphere Application Server. Make sure that all node agents within the cell are active beforehand.

   For more information, see “Enabling security” on page 69. It is important to click **Security > Global security**. Select an available realm definition from the list, and then click **Set as current** so that security is enabled upon a server restart.

   Note: In previous releases of WebSphere Application Server, the **Set as current** option is known as the **Enable global security** option.
2. Before restarting the server, log off the administrative console. You can log off by clicking **Logout** at the top menu bar.
3. Stop the server by going to the command line in the WebSphere Application Server *app_server_root*`/bin` directory and issue a `stopServer` *server_name* command.
4. Restart the server in secure mode by issuing the command `startServer` *server_name*. Once the server is secure, you cannot stop the server again without specifying an administrative user name and password. To stop the server once security is enabled, issue the command, `stopServer` *server_name* `-username` *user_id* `-password` *password*. Alternatively, you can edit the `soap.client.props` file in the

profile_root/properties directory, and edit the com.ibm.SOAP.loginUserid or com.ibm.SOAP.loginPassword properties to contain these administrative IDs.

If you have any problems restarting the server, review the output logs in the *profile_root*/logs/server_name directory. Check the Chapter 13, “Troubleshooting security configurations,” on page 989 article for any common problems.

Global security settings

Use this panel to configure administration and the default application security policy. This security configuration applies to the security policy for all administrative functions and is used as a default security policy for user applications. Security domains can be defined to override and customize the security policies for user applications.

To view this administrative console page, click **Security > Global security**.

Security has some performance impacts on your applications. The performance impacts can vary depending upon the application workload characteristics. You must first determine that the needed level of security is enabled for your applications, and then measure the impact of security on the performance of your applications.

When security is configured, validate any changes to the user registry or authentication mechanism panels. Click **Apply** to validate the user registry settings. An attempt is made to authenticate the server ID or to validate the admin ID (if internalServerID is used) to the configured user registry. Validating the user registry settings after enabling administrative security can avoid problems when you restart the server for the first time.

Security configuration wizard:

Launches a wizard that enables you to configure the basic administrative and application security settings. This process restricts administrative tasks and applications to authorized users.

Using this wizard, you can configure application security, resource or Java 2 Connector (J2C) security, and a user registry. You can configure an existing registry and enable administrative, application, and resource security.

When you apply changes made by using the security configuration wizard, administrative security is turned on by default.

Security configuration report:

Launches a report that gathers and displays the current security settings of the application server. Information is gathered about core security settings, administrative users and groups, CORBA naming roles, and cookie protection. When multiple security domains are configured the report displays the security configuration associated with each domain.

A current limitation to the report is that it does not display application level security information. The report also does not display information on Java Message Service (JMS) security, bus security, or Web Services Security.

Enable administrative security:

Specifies whether to enable administrative security for this application server domain. Administrative security requires users to authenticate before obtaining administrative control of the application server.

For more information, see the related links for administrative roles and administrative authentication.

When enabling security, set the authentication mechanism configuration and specify a valid user ID and password (or a valid admin ID when internalServerID feature is used) in the selected registry configuration.

Note: There is a difference between the user ID (which is normally called the admin ID), which identifies administrators who manage the environment, and a server ID, which is used for server-to-server communication. You do not need to enter a server ID and password when you are using the internal server ID feature. However, optionally, you can specify a server ID and password. To specify the server ID and password, complete the following steps:

1. Click **Security > Global security**.
2. Under User accounts repository, select the repository and click **Configure**.
3. Specify the server ID and password in the Server user identity section.

Information	Value
Default:	Enabled

Enable application security:

Enables security for the applications in your environment. This type of security provides application isolation and requirements for authenticating application users

In previous releases of WebSphere Application Server, when a user enabled global security, both administrative and application security were enabled. In WebSphere Application Server Version 6.1, the previous notion of global security is split into administrative security and application security, each of which you can enable separately.

As a result of this split, WebSphere Application Server clients must know whether application security is disabled at the target server. Administrative security is enabled, by default. Application security is disabled, by default. To enable application security, you must enable administrative security. Application security is in effect only when administrative security is enabled.

Information	Value
Default:	Disabled

Use Java 2 security to restrict application access to local resources:

Specifies whether to enable or disable Java 2 security permission checking. By default, access to local resources is not restricted. You can choose to disable Java 2 security, even when application security is enabled.

When the **Use Java 2 security to restrict application access to local resources** option is enabled and if an application requires more Java 2 security permissions than are granted in the default policy, the application might fail to run properly until the required permissions are granted in either the `app.policy` file or the `was.policy` file of the application. AccessControl exceptions are generated by applications that do not have all the required permissions. See the related links for more information about Java 2 security.

Information	Value
Default:	Disabled

Warn if applications are granted custom permissions:

Specifies that during application deployment and application start, the security runtime issues a warning if applications are granted any custom permissions. Custom permissions are permissions that are defined by the user applications, not Java API permissions. Java API permissions are permissions in the `java.*` and `javax.*` packages.

The application server provides support for policy file management. A number of policy files are available in this product, some of them are static and some of them are dynamic. Dynamic policy is a template of permissions for a particular type of resource. No code base is defined and no relative code base is used in the dynamic policy template. The real code base is dynamically created from the configuration and run-time data. The `filter.policy` file contains a list of permissions that you do not want an application to have according to the J2EE 1.4 specification. For more information on permissions, see the related link about Java 2 security policy files.

Important: You cannot enable this option without enabling the **Use Java 2 security to restrict application access to local resources** option.

Information	Value
Default:	Disabled

Restrict access to resource authentication data:

Enable this option to restrict application access to sensitive Java Connector Architecture (JCA) mapping authentication data.

Consider enabling this option when both of the following conditions are true:

- Java 2 security is enforced.
- The application code is granted the accessRuntimeClasses WebSphereRuntimePermission permission in the `was.policy` file found within the application enterprise archive (EAR) file. For example, the application code is granted the permission when the following line is found in your `was.policy` file:

```
permission com.ibm.websphere.security.WebSphereRuntimePermission "accessRuntimeClasses";
```

The **Restrict access to resource authentication data** option adds fine-grained Java 2 security permission checking to the default principal mapping of the WSPrincipalMappingLoginModule implementation. You must grant explicit permission to Java 2 Platform, Enterprise Edition (J2EE) applications that use the WSPrincipalMappingLoginModule implementation directly in the Java Authentication and Authorization Service (JAAS) login when **Use Java 2 security to restrict application access to local resources** and the **Restrict access to resource authentication data** options are enabled.

Information	Value
Default:	Disabled

Current realm definition:

Specifies the current setting for the active user repository.

This field is read-only.

Available realm definitions:

Specifies the available user account repositories.

The selections appear in a drop-down list containing:

- Local operating system
- Standalone LDAP registry
- Stand-alone custom registry

Set as current:

Enables the user repository after it is configured.

LDAP or a custom user registry is required when running as a UNIX non-root user or running in a multi-node environment.

You can configure settings for one of the following user repositories:

Federated repositories
Specify this setting to manage profiles in multiple repositories under a single realm. The realm can consist of identities in:

- The file-based repository that is built into the system
- One or more external repositories
- Both the built-in, file-based repository and in one or more external repositories

Note: Only a user with administrator privileges can view the federated repositories configuration.

Local operating system

You cannot use localOS in multi-node or when running as non-root on a UNIX platform.

Standalone LDAP registry

Specify this setting to use stand-alone LDAP registry settings when users and groups reside in an external LDAP directory. When security is enabled and any of these properties change, go to the **Security > Global security** panel and click **Apply** or **OK** to validate the changes.

Note: Since multiple LDAP servers are supported, this setting does not imply one LDAP registry.

Stand-alone custom registry
Specify this setting to implement your own stand-alone custom registry that implements the com.ibm.websphere.security.UserRegistry interface. When security is enabled and any of these properties change, go to the Global security panel and click **Apply** or **OK** to validate the changes.

Information	**Value**
Default:	Disabled

Configure...:

Select to configure the global security settings.

Web and SIP security:

Under Authentication, expand Web and SIP security to view links to:

- General settings
- Single sign-on (SSO)
- SPNEGO web authentication
- Trust association

General settings:

Select to specify the settings for web authentication.

Single sign-on (SSO):

Select to specify the configuration values for single sign-on (SSO).

With SSO support, web users can authenticate once when accessing both WebSphere Application Server resources, such as HTML, JavaServer Pages (JSP) files, servlets, enterprise beans, and Lotus® Domino® resources.

SPNEGO web authentication:

Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) provides a way for web clients and the server to negotiate the web authentication protocol that is used to permit communications.

Trust association:

Select to specify the settings for the trust association. Trust association is used to connect reversed proxy servers to the application servers.

You can use the global security settings or customize the settings for a domain.

Note: The use of trust association interceptors (TAIs) for SPNEGO authentication is now deprecated. The SPNEGO web authentication panels now provide a much easier way to configure SPNEGO.

RMI/IIOP security:

Under Authentication, expand RMI/IIOP security to view links to:

- CSIv2 inbound communications
- CSIv2 outbound communications

CSIv2 inbound communications:

Select to specify authentication settings for requests that are received and transport settings for connections that are accepted by this server using the Object Management Group (OMG) Common Secure Interoperability (CSI) authentication protocol.

Authentication features include three layers of authentication that you can use simultaneously:

- **CSIv2 attribute layer**. The attribute layer might contain an identity token, which is an identity from an upstream server that already is authenticated. The identity layer has the highest priority, followed by the message layer, and then the transport layer. If a client sends all three, only the identity layer is used. The only way to use the SSL client certificate as the identity is if it is the only information that is presented during the request. The client picks up the interoperable object reference (IOR) from the namespace and reads the values from the tagged component to determine what the server needs for security.
- **CSIv2 transport layer**. The transport layer, which is the lowest layer, might contain a Secure Sockets Layer (SSL) client certificate as the identity.
- **CSIv2 message layer**. The message layer might contain a user ID and password or an authenticated token with an expiration.

CSIv2 outbound communications:

Select to specify authentication settings for requests that are sent and transport settings for connections that are initiated by the server using the Object Management Group (OMG) Common Secure Interoperability (CSI) authentication protocol.

Authentication features include three layers of authentication that you can use simultaneously:

- **CSIv2 attribute layer**. The attribute layer might contain an identity token, which is an identity from an upstream server that already is authenticated. The identity layer has the highest priority, followed by the message layer, and then the transport layer. If a client sends all three, only the identity layer is used. The only way to use the SSL client certificate as the identity is if it is the only information that is

presented during the request. The client picks up the interoperable object reference (IOR) from the namespace and reads the values from the tagged component to determine what the server needs for security.
- **CSIv2 transport layer**. The transport layer, which is the lowest layer, might contain a Secure Sockets Layer (SSL) client certificate as the identity.
- **CSIv2 message layer**. The message layer might contain a user ID and password or an authenticated token with an expiration.

Java authentication and authorization service:

Under Authentication, expand Java authentication and authorization service to view links to:
- Application logins
- System logins
- J2C authentication data

Application logins:

Select to define login configurations that are used by JAAS.

Do not remove the ClientContainer, DefaultPrincipalMapping, and WSLogin login configurations because other applications might use them. If these configurations are removed, other applications might fail.

System logins:

Select to define the JAAS login configurations that are used by system resources, including the authentication mechanism, principal mapping, and credential mapping.

J2C authentication data:

Select to specify the settings for the Java Authentication and Authorization Service (JAAS) Java 2 Connector (J2C) authentication data.

You can use the global security settings or customize the settings for a domain.

LTPA:

Select to encrypt authentication information so that the application server can send the data from one server to another in a secure manner.

The encryption of authentication information that is exchanged between servers involves the Lightweight Third-Party Authentication (LTPA) mechanism.

Kerberos and LTPA:

Select to encrypt authentication information so that the application server can send the data from one server to another in a secure manner.

The encryption of authentication information that is exchanged between servers involves the Kerberos mechanism.

Note: Kerberos must be configured before this option can be selected.

Kerberos configuration:

Select to encrypt authentication information so that the application server can send data from one server to anther in a secure manner.

The encryption of the authentication information that is exchanged between servers involves the KRB5 of LTPA mechanism.

Authentication cache settings:

Select to set your authentication cache settings.

Enable Java Authentication SPI (JASPI):

Select to enable the use of Java Authentication SPI (JASPI) authentication.

You can then click **Providers** to create or edit a JASPI authentication provider and associated authentication modules in the global security configuration.

Use realm-qualified user names:

Specifies that user names that are returned by methods, such as the getUserPrincipal() method, are qualified with the security realm in which they reside.

Security domains:

Use the Security Domain link to configure additional security configurations for user applications.

For example, if you want use a different user registry for a set of user applications than the one used at the global level, you can create a security configuration with that user registry and associate it with that set of applications. These additional security configurations can be associated with various scopes (cell, clusters/servers, SIBuses). Once the security configurations have been associated with a scope all of the user applications in that scope use this security configuration. Read about “Multiple security domains” on page 126 for more detailed information.

For each security attribute, you can use the global security settings or customize settings for the domain.

External authorization providers:

Select to specify whether to use the default authorization configuration or an external authorization provider.

The external providers must be based on the Java Authorization Contract for Containers (JACC) specification to handle the Java(TM) 2 Platform, Enterprise Edition (J2EE) authorization. Do not modify any settings on the authorization provider panels unless you have configured an external security provider as a JACC authorization provider.

Custom properties:

Select to specify name-value pairs of data, where the name is a property key and the value is a string.

Specify extent of protection wizard settings

Use this security wizard page to determine whether to enable application security and restrict access to local resources. When you use the wizard, admin security is enabled by default.

To view this security wizard page, click **Security > Global security > Security configuration wizard**.

Enable application security:

Enables security for the applications in your environment. This type of security provides application isolation and requirements for authenticating application users

In previous releases of WebSphere Application Server, when a user enabled global security, both administrative and application security were enabled. In WebSphere Application Server Version 6.1, the previous notion of global security is split into administrative security and application security, each of which you can enable separately.

As a result of this split, WebSphere Application Server clients must know whether application security is disabled at the target server. Administrative security is enabled, by default. Application security is disabled, by default. To enable application security, you must enable administrative security. Application security is in effect only when administrative security is enabled.

Information	Value
Default:	Disabled

Use Java 2 security to restrict application access to local resources:

Specifies whether to enable or disable Java 2 security permission checking. By default, access to local resources is not restricted. You can choose to disable Java 2 security, even when application security is enabled.

When the **Use Java 2 security to restrict application access to local resources** option is enabled and if an application requires more Java 2 security permissions than are granted in the default policy, the application might fail to run properly until the required permissions are granted in either the `app.policy` file or the `was.policy` file of the application. AccessControl exceptions are generated by applications that do not have all the required permissions. See the related links for more information about Java 2 security.

Information	Value
Default:	Disabled

Security custom properties

Use this page to understand the psecurity.allowCustomHTTPMethodsredefined custom properties that are related to security.

To view this administrative console page, click **Security > Global security > Custom properties**. Then click **New** to add a new custom property and its associated value.

The custom properties in this topic are set in the administrative console through the previously listed path unless otherwise stated in the description.

You can use the custom properties page to define the following security custom properties:

- “com.ibm.websphere.lookupRegistryOnProcess” on page 98
- “com.ibm.websphere.security.allowAnyLogoutExitPageHost” on page 98
- “com.ibm.websphere.security.alwaysRestoreOriginalURL” on page 98
- “com.ibm.websphere.security.auth.setDRSBootstrap” on page 108
- “com.ibm.websphere.security.config.inherit.trustedRealms” on page 99
- “com.ibm.websphere.security.console.noSSLTreePortEndpoints” on page 99
- “com.ibm.websphere.security.customLTPACookieName” on page 99
- “com.ibm.websphere.security.customSSOCookieName” on page 100
- “com.ibm.websphere.security.displayRealm” on page 101
- “com.ibm.websphere.security.disableGetTokenFromMBean” on page 101
- usec_seccustomprop.dita#com.ibm.websphere.security.enableAuditForIsCallerInRole
- “com.ibm.websphere.security.goToLoginPageWhenTAIUserNotFound” on page 101
- “com.ibm.websphere.security.InvokeTAIbeforeSSO” on page 102
- “com.ibm.websphere.security.JAASAuthData.addNodeNameSecDomain” on page 102
- “com.ibm.websphere.security.JAASAuthData.removeNodeNameGlobal” on page 102
- “com.ibm.websphere.security.krb.canonical_host” on page 102
- “com.ibm.websphere.security.ldap.logicRealm” on page 103
- “com.ibm.websphere.security.ldapSSLConnectionTimeout” on page 103
- “com.ibm.websphere.security.logoutExitPageDomainList” on page 103
- “com.ibm.websphere.security.performTAIForUnprotectedURI” on page 104
- “com.ibm.websphere.security.recoverContextWithNewKeys” on page 104
- “com.ibm.websphere.security.rsaCertificateAliasCache” on page 104
- “com.ibm.websphere.security.spnego.useBuiltInMappingToSAF” on page 104
- “com.ibm.websphere.security.strictCredentialExpirationCheck” on page 104
- “com.ibm.websphere.security.tokenFromMBeanSoapTimeout” on page 105
- “com.ibm.websphere.security.useLoggedSecurityName” on page 105
- “com.ibm.websphere.security.util.csiv2SessionCacheIdleTime” on page 105
- “com.ibm.websphere.security.util.csiv2SessionCacheLimitEnabled” on page 106
- “com.ibm.websphere.security.util.csiv2SessionCacheMaxSize” on page 106
- “com.ibm.websphere.security.util.postParamMaxCookieSize” on page 107
- “com.ibm.websphere.security.web.removeCacheOnFormLogout” on page 107
- “com.ibm.websphere.security.webAlwaysLogin” on page 107
- “com.ibm.websphere.security.useLoggedSecurityName” on page 105
- “com.ibm.ws.security.addHttpOnlyAttributeToCookies” on page 108
- “com.ibm.ws.security.allowNonAdminToSecurityXML” on page 108
- “com.ibm.ws.security.config.SupportORBConfig” on page 109
- “com.ibm.ws.security.createTokenSubjectForAsynchLogin” on page 109
- “com.ibm.ws.security.defaultLoginConfig” on page 109
- “com.ibm.ws.security.failSSODuringCushion” on page 109
- “com.ibm.ws.security.ltpa.forceSoftwareJCEProviderForLTPA” on page 109
- “com.ibm.ws.security.ssoInteropModeEnabled” on page 110
- “com.ibm.ws.security.unprotectedUserRegistryMethods” on page 110
- “com.ibm.ws.security.webChallengeIfCustomSubjectNotFound” on page 111
- “com.ibm.ws.security.webInboundLoginConfig” on page 111
- “com.ibm.ws.security.webInboundPropagationEnabled” on page 111

com.ibm.audit.report.granularity:

Use this property to specify how much auditing data is recorded for each event type. If you only need to record basic information about an event, such as who did what action to what resource, and when, setting this property to `high`, might improve your application server performance.

You can specify values of `high`, `medium`, or `low` for this property. The default value is `low`.

Table 15. Type of data that is recorded for each event type based on the setting for com.ibm.audit.report.granularity. The following table indicates the type of data that is recorded for each event type based on the setting for this property.

Event type	`high` setting	`medium` setting	`low` setting
SessionContext	sessionId	sessionId, remoteHost	sessionId, remoteHost, remoteAddr, remotePort
PropagationContext (is only reported if SAP is enabled)	firstCaller (as part of the who)	firstCaller, and if verbose mode is enabled, the callerList	firstCaller, and if verbose mode is enabled, the callerList
RegistryContext	nothing is recorded	registry type	registry type
ProcessContext	nothing is recorded	realm	realm, and domain if verbose is enabled
EventContext	creationTime	creationTime, globalInstanceId	creationTime, globalInstanceId, eventTrailId, and lastTrailId if verbose mode is enabled
DelegationContext	identityName	delegationType, and identityName	delegationType, roleName, and identityName
AuthnContext	nothing is recorded	authn type	authn type
ProviderContext	nothing is recorded	provider	provider, and providerStatus
AuthnMappingContext	mappedUserName	mappedUserName, and mappedSecurityRealm	mappedUserName, mappedSecurityRealm, and mappedSecurityDomain
AuthnTermContext	terminateReason	terminateReason	terminateReason
AccessContext	progName, action, appUserName, and resourceName	progName, action, appUserName, resourceName, registryUserName, and accessDecision	progName, action, appUserName, resourceName, registryUserName, accessDecision, resourceType, permissionsChecked, permissionsGranted, rolesChecked, and rolesGranted
PolicyContext	nothing is recorded	policyName	policyName, and policyType
KeyContext	keyLabel	keyLabel, and keyLocation	keyLabel, keyLocation, and certificateLifetime
MgmtContext	nothing is recorded	mgmtType, and mgmtCommand	mgmtType, mgmtCommand, and targetInfoAttributes

com.ibm.CSI.disablePropagationCallerList:

This property disables the caller list and does not allow the caller list to change. This property prevents the creation of multiple sessions.

This property completely disables adding a caller or host list in the propagation token. Setting this property can be a benefit when the caller or host list in the propagation token is not needed in the environment.

gotcha: If the **com.ibm.CSI.propagateFirstCallerOnly** custom property is set to true, that setting takes precedence over the setting for this property.

Information	Value
Default	false

com.ibm.CSI.propagateFirstCallerOnly:

This property limits the caller list to the first caller only, which means the caller list cannot change. Setting this property to true eliminates the potential for the creation of multiple session entries.

This property logs the first caller in the propagation token that stays on the thread when security attribute propagation is enabled. Without setting this property, all caller switches get logged, which affects performance. Typically, only the first caller is of interest.

gotcha: If the **com.ibm.CSI.disablePropagationCallerList** custom property is set to true, that setting takes precedence over the setting for this property.

Information	Value
Default	true

The default value of the com.ibm.CSI.propagateFirstCallerOnly security custom property is set to `true`. When this custom property is set to `true`, the first caller in the propagation token that stays on the thread is logged when security attribute propagation is enabled. When this property is set to `false`, all of the caller switches are logged, which can affect performance.

com.ibm.CSI.rmiInboundLoginConfig:

This property specifies the Java Authentication and Authorization Service (JAAS) login configuration that is used for Remote Method Invocation (RMI) requests that are received inbound.

By knowing the login configuration, you can plug in a custom login module that can handle specific cases for RMI logins.

Information	Value
Default	system.RMI_INBOUND

com.ibm.CSI.rmiInboundMappingConfig:

This property defines the system JAAS login configuration that is used to perform application specific principal mapping.

Information	Value
Default	None

com.ibm.CSI.rmiInboundMappingEnabled:

This property, when set to `true`, enables the application specific principal mapping capability.

Information	Value
Default	false

com.ibm.CSI.rmiOutboundLoginConfig:

This property specifies the JAAS login configuration that is used for RMI requests that are sent outbound.

Primarily, this property prepares the propagated attributes in the Subject to be sent to the target server. However, you can plug in a custom login module to perform outbound mapping.

Information	Value
Default	system.RMI_OUTBOUND

com.ibm.CSI.rmiOutboundMappingEnabled:

This property, when set to `true`, enables the original caller subject embedded in the WSSubjectWrapper object to be restored.

Information	Value
Default	false

com.ibm.CSI.supportedTargetRealms:

This property enables credentials that are authenticated in the current realm to be sent to any realm that is specified in the Trusted target realms field. The Trusted target realms field is available on the CSIv2 outbound authentication panel. This property enables those realms to perform inbound mapping of the data from the current realm.

| You should not send authentication information to an unknown realm. Thus, this property provides a way
| to specify that the alternate realms are trusted. To access the CSIv2 outbound authentication panel,
| complete the following steps:

| 1. Click **Security** > **Global security**.
| 2. Under RMI/IIOP security, click **CSIv2 outbound authentication**.

com.ibm.security.multiDomain.setNamingReadUnprotected:

This property can be set to `true` if you want the CosNamingRead role to protect all naming read operations. Setting this property to true is the equivalent of assigning the CosNamingRead role the Everyone special subject. When this property is set, any assignments made to the CosNamingRead role are ignored.

Information	Value
Default	none

com.ibm.security.useFIPS:

Specifies that Federal Information Processing Standard (FIPS) algorithms are used. The application server uses the IBMJCEFIPS cryptographic provider instead of the IBMJCE cryptographic provider.

Information	Value
Default	false

com.ibm.websphere.crypto.config.certexp.notify.fromAddress:

This security property is used to customize the “from address” of certificate expiration notification email.

The value you assign to this property should be an internet address, such as “Notification@abc-company.com”. If this property is not set, the application server uses the email fromAddress: WebSphereNotification@ibm.com.

Information	Value
Default	None

com.ibm.websphere.crypto.config.certexp.notify.textEncoding:

This security property is used to customize the text encoding character set for certificate expiration notification email.

WebSphere Application Server sends notification email for certificate expiration in either US-English or the machine default character set (if non-English locale is specified). If you want a different text encoding character set for the certificate expiration notification email, you can use this property to customize the text encoding character set.

Information	Value
Default	None

com.ibm.websphere.lookupRegistryOnProcess:

This property can be set when realm registry lookups are performed via an MBean on a remote server, and the realm is local OS security.

By default, the user registry tasks listRegistryUsers and listRegistryGroups perform lookups from the current process. In the case of Network Deployment (ND), that is the deployment manager.

When dealing with a local OS user registry, lookup should occur on the actual server where the registry resides. In an ND environment, the server could be a remote machine. To perform a lookup on the server process where the registry resides, set the com.ibm.websphere.lookupRegistryOnProcess custom property to `true`.

If com.ibm.websphere.lookupRegistryOnProcess is not set, or set to `false`, then the lookup is performed on the current process. The custom property can be set using the setAdminActiveSecuritySettings task for global security or the setAppActiveSecuritySettings task for a security domain.

com.ibm.websphere.security.allowAnyLogoutExitPageHost:

When you are using application form login and logout you can provide a URL for a custom logout page. By default, the URL must point to the host to which the request is made or to its domain. If this is not done, then a generic logout page is displayed rather than a the custom logout page. If you want to be able to point to any host, then you need to set this property in the `security.xml` file to a value of `true`. Setting this property to `true` might open your systems to URL redirect attacks.

Information	Value
Default	false

com.ibm.websphere.security.alwaysRestoreOriginalURL:

Use this property to indicate whether a cookie with the value WASReqURL is honored when the custom form login processor is used.

When this property is set to `true`, the value of WASReqURL takes precedence over the current URL, and the WASReqURL cookie is removed from subsequent requests.

When this property is set to `false`, the value of the current URL takes precedence, and the WASReqURL cookie is not removed from subsequent requests.

Information	Value
Default	false

com.ibm.websphere.security.config.inherit.trustedRealms:

This property is used to inherit the global trusted realm settings from the global security configuration in the domain.

Security configuration trusted inbound and outbound realms are not inherited by default. However, there are some cases where the configuration might want to use (inherit) the settings from the global security configuration in the domain.

The value of this property can be either `true` or `false`.

com.ibm.websphere.security.console.noSSLTreePortEndpoints:

This property is used to improve the response time for large topology configurations.

When this property is set to `true` the status of the of the SSL port endpoints does not display on the Manage endpoint security configurations page in the administrative console. Displaying the status of the SSL port endpoints sometimes makes the administrative console seem like it is no longer functioning because of a longer than expected response time.

Information	Value
Default	false

com.ibm.websphere.security.customLTPACookieName:

This property is used to customize the name of the cookies used for Lightweight Third Party Authentication (LTPA) tokens.

WebSphere Application Server Version 8 and later enables you to customize the name of the cookies used for LTPA and LTPA2 tokens. Custom cookie names allow you to logically separate authentication between Single Sign-On (SSO) domains and to enable customized authentication to a particular environment.

To take advantage of this functionality, a custom property must be set. For LTPA tokens, the custom property com.ibm.websphere.security.customLTPACookieName can be set to any valid string (special characters and spaces are not permitted) for the LTPA token cookie, and com.ibm.websphere.security.customSSOCookieName for the LTPA2 (SSO) token cookie. Each property is case-sensitive.

The value for this property is a valid string.

Note: Before you set this custom property, consider the following:

- This property, as with most custom properties, can be set at the security domain level. In this manner, a separate login can be forced between an administrative console login and an application login.

- The original default LTPAToken or LTPAToken2 cookie names are accepted and trusted by WebSphere Application Server Version 8 and later. This enables compatibility with products such as Lotus Domino and WebSphere Portal which both utilize the default cookie name.
- Setting a custom cookie name can cause an authentication failure. For example, a connection to a server that has a custom cookie property set sends this custom cookie to the browser. A subsequent connection to a server that uses either the default cookie name or a different cookie name is not able to authenticate the request via a validation of the inbound cookie.
- This property does not function properly in a mixed-cell environment. For example, a deployment manager in WebSphere Application Server Version 8 and later might be able to create custom cookies. However, a WebSphere Application Server Version 7.0 node or server existing in this same cell does not understand what to do with this cookie and subsequently rejects it.
- If you utilize a product interacting with WebSphere Application Server that generates LTPA tokens, such as Lotus Domino or WebSphere Portal, be aware that these products might not be able to handle custom LTPA cookie names. Please consult the documentation for your product regarding its handling of custom LTPA cookie names.

Note: To activate this property, a restart of WebSphere Application Server is necessary.

com.ibm.websphere.security.customSSOCookieName:

This property is used to customize the name of the cookies used for Lightweight Third Party Authentication Version 2 (LTPA2) tokens.

WebSphere Application Server Version 8 and later enables you to customize the name of the cookies used for LTPA and LTPA2 tokens. Custom cookie names allow you to logically separate authentication between Single Sign-On (SSO) domains and to enable customized authentication to a particular environment.

To take advantage of this functionality, a custom property must be set. For LTPA tokens, the custom property com.ibm.websphere.security.customLTPACookieName can be set to any valid string (special characters and spaces are not permitted) for the LTPA token cookie, and com.ibm.websphere.security.customSSOCookieName for the LTPA2 (SSO) token cookie. Each property is case-sensitive.

The value for this property is a valid string.

Note: Before you set this custom property, consider the following:

- This property, as with most custom properties, can be set at the security domain level. In this manner, a separate login can be forced between an administrative console login and an application login.
- The original default LTPAToken or LTPAToken2 cookie names are accepted and trusted by WebSphere Application Server Version 8 and later. This enables compatibility with products such as Lotus Domino and WebSphere Portal which both utilize the default cookie name.
- Setting a custom cookie name can cause an authentication failure. For example, a connection to a server that has a custom cookie property set sends this custom cookie to the browser. A subsequent connection to a server that uses either the default cookie name or a different cookie name is not able to authenticate the request via a validation of the inbound cookie.
- This property does not function properly in a mixed-cell environment. For example, a deployment manager in WebSphere Application Server Version 8 and later might be able to create custom cookies. However, a WebSphere Application Server Version 7.0 node or server existing in this same cell does not understand what to do with this cookie and subsequently rejects it.
- If you utilize a product interacting with WebSphere Application Server that generates LTPA tokens, such as Lotus Domino or WebSphere Portal, be aware that these products might not be able to handle custom LTPA cookie names. Please consult the documentation for your product regarding its handling of custom LTPA cookie names.

Note: To activate this property, a restart of WebSphere Application Server is necessary.

com.ibm.websphere.security.displayRealm:

This property specifies whether the HTTP basic authentication login window displays the realm name that is not defined in the application `web.xml` file.

Note: If the realm name is defined in the application `web.xml` file, this property is ignored.

If the realm name is not defined in the `web.xml` file, one of the following occurs:

- If the property is set to `false`, the WebSphere realm name display is Default Realm.
- If this property is set to `true`, the WebSphere realm name display is the user registry realm name for the LTPA authentication mechanism or the Kerberos realm name for the Kerberos authentication mechanism.

Important: If this property is set to `true`, and the user registry's realm name contains sensitive information, it is displayed to the user. For example, if standalone LDAP configuration is used, the LDAP server hostname and port are displayed. For LocalOS, the hostname is displayed.

Information	Value
Default	false
Type	string

com.ibm.websphere.security.disableGetTokenFromMBean:

Use this property to disable the outbound SOAP call to retrieve the subject from the originating server when Single Sign-On is enabled.

Typically, when Single Sign-On is enabled, and an inbound request needs to be authenticated, the receiving server attempts to retrieve the authentication from the originating server. The connection between the sending and receiving servers never times out during this callback process.

When this property is set to `true`, the receiving server does not attempt to authenticate the inbound request.

Information	Value
Default	false

com.ibm.websphere.security.enableAuditForIsCallerInRole:

Use this property to enable audit for the isCallerInRole method call.

If you set this property to `false`, it disables auditing for the invocation of isCallerInRole. In z/OS, SMF records are not issued for the invocation.

Information	Value
Default	true

com.ibm.websphere.security.goToLoginPageWhenTAIUserNotFound:

Use this property when the user provided by a TAI is not found in the user registry so that a login page is displayed instead of an error page.

When the user provided by a TAI is not found in the user registry, WebSphere Application Server displays an error page. To adjust this behavior, set this property to `true`. Then the login page is displayed. The default setting for this property is `false` and the normal behavior for WebSphere Application Server is to display an error page.

When this property is set to `true`, the login page is displayed.

Default	false

com.ibm.websphere.security.InvokeTAIbeforeSSO:

Default invocation order of Trust Association Interceptors (TAIs) in relation to Single Sign On (SSO) user authentication can be changed using this property. The default order is to invoke Trust Association Interceptors after SSO. This property is used to change the default order of TAI invocation with SSO. The property value is a comma (,) separated list of TAI class names to be invoked before SSO.

Information	**Value**
Default	com.ibm.ws.security.spnego.TrustAssociationInterceptorImpl
Type	string

com.ibm.websphere.security.JAASAuthData.addNodeNameSecDomain:

By default, when JAAS authentication data entries are created at the domain security level, the alias name for the entry will be in the format `aliasName`. You can enable the addition of the node name to the alias name to create the alias name, in the format `nodeName/aliasName`, for the entry, by setting the following property at the domain security level.

You can set com.ibm.websphere.security.JAASAuthData.addNodeNameSecDomain=true at the global security level, to enable the addition of the node name to the alias name of JAAS authentication data entries for all security domains.

Information	**Value**
Default	false

com.ibm.websphere.security.JAASAuthData.removeNodeNameGlobal:

By default, when JAAS authentication data entries are created at the global security level, the alias name for the entry is in the format `nodeName/aliasName`. You can disable the addition of the node name to the alias name for the entry, by setting a value of `true` for this property at the global security level.

Information	**Value**
Default	false

com.ibm.websphere.security.krb.canonical_host:

This custom property specifies whether the application server uses the canonical form of the URL/HTTP host name in authenticating a client. This property can be used for both SPNEGO TAI and SPNEGO Web.

If you set this custom property to `false`, a Kerberos ticket can contain a host name that differs from the HTTP host name header, and the application server might issue the following message:

```
CWSPN0011E: An invalid SPNEGO token has been encountered while authenticating a HttpServletRequest
```

If you set this custom property to `true`, you can avoid this error message and allow the application server to authenticate using the canonical form of the URL/HTTP host name.

Information	Value
Default	true

com.ibm.websphere.security.ldap.logicRealm:

This custom property enables you to change the name of the realm that is placed in the token.

This custom property enables you to configure each cell to have its own LDAP host for interoperability and backward compatibility. Also, it provides flexibility for adding or removing the LDAP host dynamically. If you are migrating a previous installation, this modified realm name does not take effect until administrative security is re-enabled. To be compatible with a previous release that does not support the logic realm, the name must be the same name that is used by the previous installation. You must use the LDAP host name, including a trailing colon and port number.

Information	Value
Type	String

This property must be set as the custom property of a stand-alone LDAP registry. To set this custom property, in the administrative console:

1. Click **Security > Global security**.
2. Under User account repository, expand the Available realm definitions list, and select **Standalone LDAP registry**, and then click **Configure**.
3. Under Custom properties, click **New** , and then enter `com.ibm.websphere.security.ldap.logicRealm` in the **Name** field, and the new name of the realm that is placed in the token in the **Value** field.
4. Select this custom property and then click **Apply** or **OK**.

com.ibm.websphere.security.ldapSSLConnectionTimeout:

Use this property, when SSL is enabled on the LDAP server, to specify, in milliseconds, the maximum amount of time the Java Virtual Machine (JVM) waits for a socket connection before issuing a timeout.

If one or more standalone LDAP servers are offline when a server process starts, and LDAP-SSL is enabled, there might be a delay of up to three minutes in the startup procedure, even if you specify a value for the com.sun.jndi.ldap.connect.timeout custom property. When LDAP-SSL is enabled, any value specified for the com.sun.jndi.ldap.connect.timeout property is ignored.

When a value is specified for this property, the JVM tries to use this connection timeout value when attempting to complete a socket connection, instead of trying to establish a directory context. When no value is specified for this property, the JVM tries to establish a directory context.

There is no default value for this property.

com.ibm.websphere.security.logoutExitPageDomainList:

When you are using application form login and logout, you can provide a URL for a custom logout page. By default, the URL must point to the host to which the request is made or to its domain. If this is not done, then a generic logout page is displayed rather than a the custom logout page. If you need to point to a different host, then you can populate this property in the `security.xml` file with a pipe (|) separated list of URLs that are allowed for the logout page.

Information	Value
Default	none

com.ibm.websphere.security.performTAIForUnprotectedURI:

This property is used to specify TAI invocation behavior when **Use available authentication data when an unprotected URI is accessed** is selected in the administrative console.

Information	Value
Default	false

Note: In previous versions of WebSphere Application Server, the default value of this custom property was `true`. For WebSphere Application Server Version 8.0.0.1, the default value is now `false`.

com.ibm.websphere.security.recoverContextWithNewKeys:

This property affects behavior when deserializing a security context that was previously saved as part of asynchronous security processing for Web Services or Asynch Beans.

When this property is set to `true`, the security context can be de-serialized even when the LTPA keys have changed since the context was serialized out. This property should be set to `true` if the security context deserialization fails with a WSSecurityException containing this message: `Validation of LTPA token failed due to invalid keys or token type`.

Information	Value
Default	false

com.ibm.websphere.security.rsaCertificateAliasCache:

This property is used to control the size of the alias cache.

The default value is 5000 and can be increased for larger deployments. You do not need to add this property unless your Job Manager topology exceeds 5000 registered nodes.

The value must be entered into the range of 1 - N, where `N` is a valid positive integer that is greater than or equal to the number of nodes registered with the Job Manager.

Information	Value
Default	5000

com.ibm.websphere.security.spnego.useBuiltInMappingToSAF:

This property is used to ensure that a mapping from a Kerberos principal to a RACF ID is performed for SPNEGO web authentication.

If you do not add this property to your security settings, and set it to `true`, a mapping from a Kerberos principal to a RACF ID is not performed for SPNEGO web authentication.

gotcha: If Kerberos authentication is used in combination with SPNEGO Web authentication, configuring a built-in mapping for either Kerberos or SPNEGO results in a mapping being done for both.

Information	Value
Default	false

com.ibm.websphere.security.strictCredentialExpirationCheck:

Specifies whether credential expiration check occurs for a local Enterprise JavaBeans (EJB) call. Typically, when an EJB invokes another EJB that is located in a local machine, a direct method invocation occurs even if the credentials of the original invoker expire before the local EJB call occurs.

If this property is set to `true`, a credential expiration check occurs on a local EJB call before the EJB is invoked on the local machine. If the credentials have expired, the EJB call is rejected.

If this property is set to `false`, a credential expiration check does not occur for a local EJB call.

Information	Value
Default	false

com.ibm.websphere.security.tokenFromMBeanSoapTimeout:

Use this property to specify the amount of time the receiving server waits for an outbound SOAP call to retrieve the proper authentication from the originating server when Single Sign-On is enabled.

There is no default value for this property. If no value is specified, the global SOAP timeout value is used as the timeout value for the SOAP connection.

com.ibm.websphere.security.useLoggedSecurityName:

This is a custom property of user registries. This property alters the behavior of creating WSCredential.

A setting of `false` indicates that the security name returned by a user registry is always used to construct WSCredential.

A setting of `true` indicates that either a security name that is supplied by login module is used or a display name that was supplied by a user registry is used. This setting is compatible with WebSphere Application Server Version 6.1 and earlier.

Information	Value
Default	false

com.ibm.websphere.security.util.csiv2SessionCacheIdleTime:

This property specifies the time in milliseconds that a CSIv2 session can remain idle before being deleted. The session is deleted if the com.ibm.websphere.security.util.csiv2SessionCacheLimitEnabled custom property is set to `true`, and the maximum size of the CSIv2 session cache is exceeded.

This custom property only applies if you enable stateful sessions, set the com.ibm.websphere.security.util.csiv2SessionCacheLimitEnabled custom property to true, and set a value for the com.ibm.websphere.security.util.csiv2SessionCacheMaxSize custom property. Consider decreasing the value for this custom property if your environment uses Kerberos authentication and has a short clock skew for the configured key distribution center (KDC). In this scenario, a short clock skew is defined as less than 20 minutes.

Important: Do not set a value for this function through the custom property panel because the value is not validated against the expected range of values. Instead, set the value on the CSIv2 outbound communications panel, which is available in the administrative console by completing the following steps:

1. Expand the **Security** section and click **Global security**.
2. Expand the **RMI/IIOP security** section and click **CSIv2 outbound communications**

You can set the value in the **Idle session timeout** field. However, when you specify this value on the CSIv2 outbound communications panel, the administrative console value is expected in seconds and not milliseconds.

The range of values for this custom property is 60,000 to 86,400,000 milliseconds. By default, the value is not set.

com.ibm.websphere.security.util.csiv2SessionCacheLimitEnabled:

This custom property specifies whether to limit the size of the CSIv2 session cache.

When you set this custom property value to `true`, you must set values for the com.ibm.websphere.security.util.csiv2SessionCacheIdleTime and com.ibm.websphere.security.util.csiv2SessionCacheMaxSize custom properties. When you set this custom property to `false`, the CSIv2 session cache is not limited. The default property value is false.

Consider setting this custom property to `true` if your environment uses Kerberos authentication and has a small clock skew for the configured key distribution center (KDC). In this scenario, a small clock skew is defined as less than 20 minutes. A small clock skew can result in a larger number of rejected CSIv2 sessions. However, with a smaller value for the com.ibm.websphere.security.util.csiv2SessionCacheIdleTime custom property, the application server can clean out these rejected sessions more frequently and potentially reduce the resource shortages.

Important: This custom property only applies if you enable the stateful sessions.

Important: Although you can enable the CSIv2 session cache limit option as a custom property, it is advisable that you enable the option on the CSIv2 outbound communications panel, which is available in the administrative console by completing the following steps:

1. Expand the **Security** section and click **Global security**.
2. Expand the **RMI/IIOP security** section and click **CSIv2 outbound communications**

You can enable the **Enable CSIv2 session cache limit** option. The default value is false.

com.ibm.websphere.security.util.csiv2SessionCacheMaxSize:

This property specifies the maximum size of the session cache after which expired sessions are deleted from the cache.

Expired sessions are defined as sessions that are idle longer than the time that is specified by the com.ibm.websphere.security.util.csiv2SessionCacheIdleTime custom property. When you use the com.ibm.websphere.security.util.csiv2SessionCacheMaxSize custom property, consider setting its value between 100 and 1000 entries.

Consider specifying a value for this custom property if your environment uses Kerberos authentication and has a small clock skew for the configured key distribution center (KDC). In this scenario, a small clock skew is defined as less than 20 minutes. Consider increasing the value of this custom property if the small cache size causes the garbage collection to run so frequently that it impacts the performance of the application server.

This custom property only applies if you enable stateful sessions, set the com.ibm.websphere.security.util.csiv2SessionCacheLimitEnabled custom property to true, and set a value for the com.ibm.websphere.security.util.csiv2SessionCacheIdleTime custom property.

Important: Do not set a value for this function through the custom property panel because the value is not validated against the expected range of values. Instead, set the value on the CSIv2 outbound communications panel, which is available in the administrative console by completing the following steps:

1. Expand the **Security** section and click **Global security**.
2. Expand the **RMI/IIOP security** section and click **CSIv2 outbound communications**

You can set the value in the **Maximum cache size** field.

The range of values for this custom property is 100 to 1000 entries. By default, the value is not set.

com.ibm.websphere.security.util.postParamMaxCookieSize:

This property sets a size limit for WASPostParam cookies being generated by the security code.

When the Use available authentication data when an unprotected URI is accessed option is enabled and Form-based authentication is being used this, a WASPOSTParam is generated during the authentication procedure of the HTTP POST request even if the target URL is unprotected. A WASPOSTParam cookie is a temporary cookie used to store HTTP POST parameters. This results in the Web client being sent the unnecessary cookie with an HTTP response. This might cause unexpected behavior when the size of the cookie is larger than the browser limit. To avoid this behavior, com.ibm.websphere.security.util.postParamMaxCookieSize can be set to cause the security code to stop generating the cookie if the maximum size specified by this property is reached. The value of this property must be a positive integer and represents the maximum size of the cookie in bytes.

Information	Value
Default	none

com.ibm.websphere.security.web.removeCacheOnFormLogout:

This custom property enables you to specify whether a cached object is removed from the authentication cache and the dynamic cache when a form logout occurs. A form logout is a mechanism that enables a user to log out of an application without having to close all Web-browser sessions.

When this property is set to false, corresponding cached entries are not removed from the authentication cache and the dynamic cache when a form logout occurs. As a result, if the same user logs back in after a form logout, the cached object is reused.

gotcha: Because the original cached object was created during a previous login session, the expiration time for the object might be shorter than the configured timeout value.

When this property is set to true, the cached entries are removed from the authentication cache and the dynamic cache when a form logout occurs.

The default value is true.

com.ibm.websphere.security.webAlwaysLogin:

This property specifies whether the login() method will throw an exception if an identity had already been authenticated. You can overwrite this behavior by setting this property to `true`.

Information	Value
Default	false
Type	string

Note: The login() method always uses the user ID and password to authenticate to the WebSphere application server irrespective of the presence of the SSO information in the HttpServletRequest.

com.ibm.ws.security.addHttpOnlyAttributeToCookies:

This custom property enables you to set the HTTPOnly attribute for single sign-on (SSO) cookies.

You can use the com.ibm.ws.security.addHttpOnlyAttributeToCookies custom property to protect cookies that contain sensitive values. When you set this custom property value to `true`, the application server sets the HTTPOnly attribute for SSO cookies whose values are set by the server. The HTTPOnly attribute enables the protection of sensitive values in cookies.

Also, a true value enables the application server to properly recognize, accept, and process inbound cookies with HTTPOnly attributes and inhibit any *cross-site scripting* from accessing sensitive cookie information.

A common security problem, which impacts web servers, is cross-site scripting. Cross-site scripting is a server-side vulnerability that is often created when user input is rendered as HTML. Cross-site scripting attacks can expose sensitive information about the users of the website. Most modern web browsers honor the HTTPOnly attribute to prevent this attack. A cookie with this attribute is called an *HTTPOnly cookie*. Information that exists in an HTTPOnly cookie is less likely to be disclosed to a hacker or a malicious website. For more information about the HTTPOnly attribute, see the Open Web Application Security Project (OWASP) website.

Important: When you use this custom property, HTTPOnly attribute is not added to every cookie that passes through the application server. Also, the attribute is not added to other non-secure cookies that are created by the application server. A list of non-HTTPOnly cookies includes:

- JSESSIONID cookies
- SSO cookies that are created by authenticators or providers from another software vendor
- Client or browser cookies that do not already contain the HTTPOnly attribute

You can set or remove this custom property from the Single sign-on panel in the administrative console by doing the following:

1. Click **Security** > **Global security**.
2. Under Authentication, click **Web and SIP security** > **Single sign-on (SSO)**.

Information	Value
Default	true
Type	Boolean

com.ibm.ws.security.allowNonAdminToSecurityXML:

This property specifies whether the non-admin security roles are allowed to modify the `security.xml` file. Setting this property to `true` gives non-admin security roles the ability to modify the `security.xml file`. In Version 6.1 and later, by default, non-admin security roles have the ability to modify the `security.xml` file.

Information	Value
Default	false
Type	Boolean

com.ibm.websphere.security.auth.setDRSBootstrap:

Specifies whether the data replication service (DRS) enables the DRSbootstrap function.

In high volume environments, dynamic cache data replication might increase the amount of time that it takes a server to start. If you experience slow server startups because of data replication, add this property to your server security settings and set it to `false`. When is property is set to false, the data replication service disables the DRSbootstrap function.

True is the default setting for this property.

com.ibm.ws.security.config.SupportORBConfig:

Specifies whether to check or not check the object request broker (ORB) for properties. This property needs to be set as a system property. You set this property to `true` or `yes` so that the ORB is checked for properties. For any other setting, the ORB is completely ignored.

The property is to be used when a pluggable application client connects to the WebSphere Application Server. Specifically, this property is used whenever a hashmap containing security properties is passed in a hashmap on a new InitialContext(env) call.

com.ibm.ws.security.createTokenSubjectForAsynchLogin:

In this release, the actual LTPA token data is not available from a WSCredential.getCredentialToken() call when called from an asynchronous bean. For an existing configuration, you can add the `com.ibm.ws.security.createTokenSubjectForAsynchLogin` custom property and a `true` value to allow the LTPAToken to be forwarded to asynchronous beans. This property allows portlets to successfully perform LTPA token forwarding. This custom property is case sensitive. You must restart the application server after you add this custom property.

gotcha: This custom property applies only to system conditions where Server A makes EJB calls from asynchronous beans to Server B. This property does not apply for JAAS login situations.

Information	Value
Default	not applicable

com.ibm.ws.security.defaultLoginConfig:

This property is the JAAS login configuration that is used for logins that do not fall under the WEB_INBOUND, RMI_OUTBOUND, or RMI_INBOUND login configuration categories.

Internal authentication and protocols that do not have specific JAAS plug points call the system login configuration that is referenced by com.ibm.ws.security.defaultLoginConfig configuration.

Information	Value
Default	system.DEFAULT

com.ibm.ws.security.failSSODuringCushion:

Use the com.ibm.ws.security.failSSODuringCushion custom property to update custom JAAS Subject data for the LTPA token.

When you do not set this custom property to `true`, new JAAS Subjects might not contain the custom JAAS Subject data.

The default value is true.

com.ibm.ws.security.ltpa.forceSoftwareJCEProviderForLTPA:

Use the com.ibm.ws.security.ltpa.forceSoftwareJCEProviderForLTPA custom property to correct an “invalid library name” error when you attempt to use a PKCS11 type keystore with a Java client.

The `ssl.client.props` file points to a configuration file, which in turn, points to the library name for the cryptographic device. The code for the Java client looks for a keystore type for the correct provider name. Without this custom property, the keystore type constant for PKCS11 is not specified correctly as it references the IBMPKCS11Impl provider instead. Also, the Lightweight Third Party Authentication (LTPA) code uses the provider list to determine the Java Cryptography Extension (JCE) provider. This approach causes a problem when Secure Sockets Layer (SSL) acceleration is attempted because the IBMPKCS11Impl provider needs to be listed before the IBMJCE provider within the `java.security` file.

This custom property corrects both issues so that SSL and other cryptographic mechanisms can use hardware acceleration.

Note: LTPA cannot use hardware acceleration because the software keys for LTPA do not implement the java.security.interfaces.RSAPrivateCrtKey interface, which is required by many accelerator cards.

Set this custom property to `true` when you want to use a PKCS11 type keystore with a Java client.

Information	Value
Default	false

com.ibm.ws.security.ltpa.useCRT:

Use this property to improve the CPU utilization during the sign() operation that occurs when a new LTPA2 (SSO) token is created. When this property is set to `true`, the product implements the Chinese Remainder Theorem (CRT) algorithm when signing the new token. This property has no effect on the old style LTPA token.

Information	Value
Default	false

com.ibm.ws.security.ssoInteropModeEnabled:

This property determines whether to send LtpaToken2 and LtpaToken cookies in the response to a web request (interoperable).

When this property value is false, the application server just sends the new LtpaToken2 cookie which is stronger, but not interoperable with some other products and WebSphere Application Server releases prior to Version 5.1.1. In most cases, the old LtpaToken cookie is not needed and you can set this property to `false`.

Information	Value
Default	true

com.ibm.ws.security.unprotectedUserRegistryMethods:

Specifies the method names on the UserRegistry interface, such as getRealm, getUsers, and isValidUser, that you do not want protected from remote access. If you specify multiple method names, separate the names with either a space, a comma, a semi-colon, and a separator bar. See your implementation of the UserRegistry interface file for a complete list of valid method names.

If you specify an `*` as the value for this property, all methods are unprotected from remote access. If a value is not specified for this property, all methods are protected from remote access.

If an attempt is made to remotely access a protected UserRegistry interface method, the remote process receives a CORBA NO_PERMISSION exception with minor code 49421098.

There is no default value for this property.

com.ibm.ws.security.webChallengeIfCustomSubjectNotFound:

This property determines the behavior of a single sign-on LtpaToken2 login.

If the token contains a custom cache key and the custom Subject cannot be found, then the token is used to log in directly as the custom information needs to be regathered if this property value is set to `true`. A challenge also occurs so that the user is required to login again. When this property value is set to `false` and the custom Subject is not found, the LtpaToken2 is used to login and gather all of the registry attributes. However, the token might not obtain any of the special attributes that downstream applications might expect.

Information	**Value**
Default	true

com.ibm.ws.security.webInboundLoginConfig:

This property is the JAAS login configuration that is used for web requests that are received inbound.

By knowing the login configuration, you can plug in a custom login module that can handle specific cases for web logins.

Information	**Value**
Default	system.WEB_INBOUND

com.ibm.ws.security.webInboundPropagationEnabled:

This property determines whether a received LtpaToken2 cookie should search for the propagated attributes locally before searching the original login server that is specified in the token. After the propagated attributes are received, the Subject is regenerated and the custom attributes are preserved.

Information	**Value**
Default	true

com.ibm.wsspi.security.cred.refreshGroups:

This property affects behavior when deserializing a security context that was previously saved as part of asynchronous security processing for Web Services or Asynch Beans.

When this property is set to `true`, the user registry is accessed to get the groups associated with the user. If the user still exists in the registry, the groups from the user registry are used instead of the groups that were serialized in the security context. If the user is not found in the user registry, and the verifyUser property is set to `false`, the groups from the security context are used.

Information	**Value**
Default	false

com.ibm.wsspi.security.cred.verifyUser:

This property affects behavior when deserializing a security context that was previously saved as part of asynchronous security processing for Web Services or Asynch Beans.

When this property is set to `true`, the user registry is accessed to verify that the user from the security context still exists. If it does not exist, a WSLoginFailedException is thrown.

Information	Value
Default	false

com.ibm.wsspi.security.ltpa.tokenFactory:

This property specifies the Lightweight Third Party Authentication (LTPA) token factories that can be used to validate the LTPA tokens.

Validation occurs in the order in which the token factories are specified because LTPA tokens do not have object identifiers (OIDs) that specify the token type. The Application Server validates the tokens using each token factory until validation is successful. The order that is specified for this property is the most likely order of the received tokens. Specify multiple token factories by separating them with a pipe (|) without spaces before or following the pipe.

Information	Value
Default	com.ibm.ws.security.ltpa.LTPATokenFactory | com.ibm.ws.security.ltpa.LTPAToken2Factory | com.ibm.ws.security.ltpa.AuthzPropTokenFactory

com.ibm.wsspi.security.token.authenticationTokenFactory:

This property specifies the implementation that is used for an authentication token in the attribute propagation framework. The property provides an old LTPA token implementation for use as the authentication token.

Information	Value
Default	com.ibm.ws.security.ltpa.LTPATokenFactory

com.ibm.wsspi.security.token.authorizationTokenFactory:

This property specifies the implementation that is used for an authorization token. This token factory encodes the authorization information.

Information	Value
Default	com.ibm.ws.security.ltpa.AuthzPropTokenFactory

com.ibm.wsspi.security.token.propagationTokenFactory:

This property specifies the implementation that is used for a propagation token. This token factory encodes the propagation token information.

The propagation token is on the thread of execution and is not associated with any specific user Subjects. The token follows the invocation downstream flow wherever the process leads.

Information	Value
Default	com.ibm.ws.security.ltpa.AuthzPropTokenFactory

com.ibm.wsspi.security.token.singleSignonTokenFactory:

This property specifies the implementation that is used for a Single Sign-on (SSO) token. This implementation is the cookie that is set when propagation is enabled regardless of the state of the com.ibm.ws.security.ssoInteropModeEnabled property.

By default, this implementation is the LtpaToken2 cookie.

Information	Value
Default	com.ibm.ws.security.ltpa.LTPAToken2Factory

com.ibm.wsspi.wssecurity.kerberos.failAuthForExpiredKerberosToken:

Use this property to specify how you want the system to handle authentication for a request after the Kerberos token for the request expires.

When this property is set to `true`, if a Kerberos token cannot be refreshed after it expires, authentication for the request fails.

When this property is set to `false`, authentication for the request does not fail even if the token has expired.

The default value for this property is false.

security.allowCustomHTTPMethods:

Use this custom property to permit custom HTTP methods. The custom HTTP methods are other than the standard HTTP methods, which are: DELETE, GET, HEAD, OPTIONS, POST, PUT or TRACE.

When this property is set to `false`, which is the default, if a combination of a URI pattern and a custom HTTP method are not listed in the security-constraint element, a search of the security constraint is performed using an URI pattern only. If there is a match, the value of the <auth-constraints> element is enforced. This behavior minimizes a potential security exposure.

When this property is set to `true`, the custom HTTP methods are treated as the standard HTTP methods. An authorization decision is made by both the URI pattern and the HTTP method. To properly protect a target URI, make sure that the proper HTTP methods are listed in the <web-resource-collection> element.

security.enablePluggableAuthentication:

This property is no longer used. Instead, use WEB_INBOUND login configuration.

Complete the following steps to modify the WEB_INBOUND login configuration:

1. Click **Security** > **Global security**.
2. Under Java Authentication and Authorization Service, click `System logins`.

Information	Value
Default	true

security.useDefaultPolicyWhenJ2SDisabled:

The NullDynamicPolicy.getPermissions method provides an option to delegate a default policy class to construct a Permissions object when this property is set to `true`. When this property is set to `false`, an empty Permissions object is returned.

Information	Value
Default	false

Security custom property collection

Use this page to view and manage arbitrary name-value pairs of data, where the name is a property key and the value is a string value that can be used to set internal system configuration properties.

The administrative console contains several custom properties pages that work similarly. To view one of these administrative pages, click a **Custom properties** link.

Name:

Specifies the name (or key) for the property.

Each property name must be unique. If the same name is used for multiple properties, the value specified for the first property is used.

Do not start your property names with `was.` because this prefix is reserved for properties that are predefined in the application server.

Value:

Specifies the value paired with the specified name.

Description:

Provides information about the name-value pair.

Security custom property settings

Use this page to configure arbitrary name-value pairs of data, where the name is a property key and the value is a string value that can be used to set internal system configuration properties. Defining a new property enables you to configure a setting beyond that which is available in the administrative console.

The administrative console contains several custom property settings pages that work similarly. To view one of these administrative pages, click **Custom properties**.

Name:

Specifies the name (or key) for the property.

Each property name must be unique. If the same name is used for multiple properties, the value specified for the first property is used.

Do not start your property names with `was.` because this prefix is reserved for properties that are predefined in the product.

Information	**Value**
Data type	String

Value:

Specifies the value paired with the specified name.

Information	**Value**
Data type	String

Description:

Provides information about the name and value pair.

Information	Value
Data type	String

Testing security after enabling it

Basic tests are available that show whether the fundamental security components are working properly. Use this task to validate your security configuration.

Before you begin

After configuring administrative security and restarting all of your servers in a secure mode, validate that security is properly enabled.

There are a few techniques that you can use to test the various security login types. For example, you can test the Web-based BasicAuth login, Web-based form login, and the Java client BasicAuth login.

Basic tests are available that show whether the fundamental security components are working properly. Complete the following steps to validate your security configuration:

Procedure

1. After enabling security, verify that your system comes up in secure mode.
2. Test the Web-based BasicAuth with *Snoop*, by accessing the following URL: `http://hostname.domain:9080/snoop`.

   A login panel is displayed. If a login panel does not display, then a problem exists. If the panel appears, type in any valid user ID and password in your configured user registry.
3. Test the Web-based form login by starting the administrative console: `http://hostname.domain:`*`port_number`*`/ibm/console`. A form-based login page is displayed. If a login page does not appear, try accessing the administrative console by typing `https://myhost.domain:9043/ibm/console`.

   Type in the administrative user ID and password that are used for configuring your user registry when configuring security.
4. Test Java Client BasicAuth with *dumpNameSpace*.

   Use the *`app_server_root`*`/bin/dumpNameSpace.bat` file. A login panel appears. If a login panel does not appear, there is a problem. Type in any valid user ID and password in your configured user registry.
5. Test all of your applications in secure mode.
6. If all the tests pass, proceed with more rigorous testing of your secured applications. If you have any problems, review the SYSOUT and SYSPRINT logs. For more information on common problems, see Chapter 13, “Troubleshooting security configurations,” on page 989.

Results

The results of these tests, if successful, indicate that security is fully enabled and working properly.

Security Configuration Wizard

The Security Configuration Wizard guides you through the process of completing the basic requirements to secure your application serving environment.

This wizard is available from the Security menu from the navigation pane of the admin console. To get to the wizard, navigate to **Security** > **Global security** > **Security Configuration Wizard**.

Step one of the configuration wizard allows you to choose the level of security desired. Application-level security is selected by default. You also have the option of selecting Java 2 security.

Step two of the configuration wizard allows you to select a user repository. You have the following options:

- "Federated repository wizard settings" on page 237
- "Local operating system wizard settings" on page 170
- "Stand-alone custom registry wizard settings" on page 201
- "Standalone LDAP registry wizard settings" on page 177

Step three of the configuration wizard allows you to specify the local operating system user and group definitions as the repository, and, if necessary, to provide the name of a user with administrator privileges.

Step four of the configuration wizard provides a summary of the results of the configuration process.

Security configuration report

The security configuration report gathers and displays the current security settings of the application server. Information is gathered about core security settings, administrative users and groups, CORBA naming roles, and cookie protection. When multiple security domains are configured, each security domain has it's own report with a subset of the sections shown in the global security report that apply to the domain.

The security configuration report now includes information about session security, web Attributes, and the HttpOnly setting to enable you to get a more complete view of your server security settings.

The report is a table with four columns: **Console Name**, **Security Configuration Name**, **Value** and **Console Path Name**. The security information gathered is divided into sections, and groups common security information. A row highlighted in blue with a title in the first column starts a new section.

The Security Configuration Report can be run from the administrative console by selecting **Security** > **Global Security** and then clicking **Security Configuration Report**. A new window displays the report information.

The columns

Console Name
Contains the name of the security attribute as found in the administrative console. If the value in this column is on a row highlighted in blue, and is the only entry on the row, then it is the start of a new section.

Security Configuration Name
Contains the security attribute as found in the configuration file.

Value Contains the value of the security attribute.

Console Path Name
Contains the path where the attribute is found on the console.

The sections

Security Settings
Displays information about the top-level security attributes. These attributes set the default for administrative security for the server, such as whether security is enabled, the default user registry, or if Java security is enabled.

For more information, read the Global security settings article.

Authentication Mechanisms and expirations
Contains all the attributes associated with each authentication mechanisms and trust associations as defined in the configuration.

User Registry
Displays the attributes for the default user registry for the server.

Authorization configuration
Displays attributes configured for an external Java Authorization Contract for Containers (JACC) provider.

Application login configuration
Displays application JAAS login entries and their login modules attributes.

CSI Displays the attributes that define the inbound and outbound information for the Common Secure Interoperability (CSI) protocol.

SSL configuration repertoires
Displays the attributes that make up the Secure Sockets Layer (SSL) configuration used by the server. There can be multiple SSL configurations defined, and information about each is displayed. This object is often referenced by an SSL configuration group object used to associate it with an inbound or an outbound connection.

For more information, read the SSL configurations collection article.

Key stores
Displays the keystore attributes for each keystore in the configuration. Keystore objects in the configuration are often referenced by an SSL configuration object in the configuration.

For more information, read the Personal certificates collection article.

Trust managers
Displays the attributes that make up trust managers that can be used by the server. Trust manager objects in the configuration are typically referenced by an SSL configuration object.

For the more information, read the Trust managers collection article.

Key managers
Displays the attributes that make up the key managers that are used by the server. Key manager objects in the configuration are typically referenced by an SSL configuration object.

For more information, read the Key managers collection article.

SSL configuration group
Displays the attributes that make up an SSL configuration that are used for an outbound or an inbound connection.

Management scope
Displays the attributes that make up a management scope. The SSL configuration-related objects in the security configuration are defined within a management scope to reference the management scope object.

For more information, read the Management scope configurations article.

Key set groups
Displays the attributes that make up a group of key sets, which are used to manage public, private and shared keys.

For more information, read the Key set groups collection article.

Key set
Displays the attributes that make up the key set, which is used to manage public, private, and shared keys.

For more information, read the Key sets collection article.

Schedules
Displays the attributes that make up the scheduled process in the security configuration.

Notifications
Displays the attributes that make up notification objects in the security configuration.

Manage certificate expiration
Displays the attributes that define how startCertificateExpMonitor is run on the server.

System login configuration
Displays the attributes that define the System login entries and their login modules.

For more information, read the System login configuration entry settings for Java Authentication and Authorization Service article.

Custom properties
Displays all the custom properties that are defined in the security configuration.

For more information, read the Custom properties article.

Web Authentication
Displays properties that are used to define web authentication used by the server.

For more information, read the web authentication settings article.

Administrative Users and Groups
Displays the attributes that define roles and the users and groups associated with them as found in the `admin-authz.xml` file. The column titled **`Administrative Role Name`** contains the name of the administrative role. A column titled **`Administrative Role Value`** contains the user ID associated with the role (if one exists).

For more information, read the Administrative roles article.

Corba Naming Console Names
Displays the defined CORBA naming roles and the users that are assigned to the roles.

For more information, read the Administrative group roles and CORBA naming service groups article.

Console Name for Certificate Management
Lists all the certificate in keystore that are defined in the security configuration. There is also information about the certificates location and their validity period.

Cookie Protection
Displays attributes that pertain to HTTP Cookies. This section differs from other sections since information is gathered from different configuration files. The HttpOnly custom property, the web authentication com.ibm.wsspi.security.web.webAuthReq property, and the session security setting on each server are displayed on the report.

Java Authorization SPI Configuration
Displays the attributes that are defined for the Java Authorization SPI (JASPI) configuration. If there is a JASPI configuration object in the security configuration, information is included concerning whether JASPI is enabled, the name of the default JASPI provider, and a list of defined providers and their authentication modules.

Note: If JASPI has not been configured, this section is not shown in the security configuration report.

Adding a new custom property in a global security configuration or in a security domain configuration

Custom properties are arbitrary name-value pairs of data, where the name is a property key and the value is a string value that can be used to set internal system configuration properties. Defining a new property

enables you to configure settings beyond those that are available in the administrative console. You can add new security custom properties in a security configuration or in a security domain configuration.

About this task

Adding a new custom property in a global security configuration using the administrative console

1. Click **Security > Global security > Custom properties**.
2. Click **New**,
3. Enter the property key name in the **Name** field.

   Each property key name must be unique. If the same name is used for multiple properties, the value specified for the first property is used.

   Do not start your property names with `was`, because this prefix is reserved for properties that are predefined in the application server.
4. Enter the property value in the **Value** field.
5. Click **Apply** or **Save**.

You can also use the -customProperties flag in the setAdminActiveSecuritySettings wsadmin command to add a new custom property in a global security configuration. See the SecurityConfigurationCommands command group for the AdminTask object article for more information about this command. For example:

```
wsadmin>AdminTask.setAdminActiveSecuritySettings('[-customProperties
["com.ibm.websphere.security.test=false"]]')
```

Adding a new custom property in a security domain configuration using the administrative console

1. Click **Security > Security domains**.
2. Select the global security domain you want to add a new custom property to.
3. Click **Custom properties**.
4. Click **New**.
5. Enter the property key name in the **Name** field.

   Each property key name must be unique. If the same name is used for multiple properties, the value specified for the first property is used.

   Do not start your property names with `was`, because this prefix is reserved for properties that are predefined in the application server.
6. Enter the property value in the **Value** field
7. Click **Apply** or **Save**.

You can also use the -customProperties flag in the setAppActiveSecuritySettings wsadmin command to add a new custom property in a global security domain configuration. See the SecurityConfigurationCommands command group for the AdminTask object article for more information about this command. Use the -securityDomainName flag to specify the security domain where the custom property is located. For example:

```
wsadmin>AdminTask.setAppActiveSecuritySettings('[ -securityDomainName testDomain
-customProperties ["com.ibm.websphere.security.test=false"]]')
```

Modifying an existing custom property in a global security configuration or in a security domain configuration

Custom properties are arbitrary name-value pairs of data, where the name is a property key and the value is a string value that can be used to set internal system configuration properties. Defining a new property enables you to configure settings beyond those that are available in the administrative console. You can modify existing security custom properties in a global security configuration or in a security domain configuration.

About this task

Modifying an existing custom property in a global security configuration using the administrative console

1. Click **Security > Global security > Custom properies**.
2. Select the custom property you want to modify.
3. Click **Edit** In the **Value** field, and then enter the value you want to modify.
4. Click **Apply** or **Save**.

You can also use the -customProperties flag in the setAdminActiveSecuritySettings wsadmin command to modify an existing custom property in a global security configuration. See the SecurityConfigurationCommands command group for the AdminTask object article for more information about this command. For example:

```
wsadmin>AdminTask.setAdminActiveSecuritySettings('[-customProperties
["com.ibm.websphere.security.test=false"]]')
```

Modifying an existing custom property in a security domain configuration using the administrative console

1. Click **Security > Security domains**.
2. Select the global security domain you want to modify.
3. Click **Custom properties**.
4. Select the custom property you want to modify.
5. Click **Edit**.In the **Value** field, and then enter the value you want to modify.
6. Click **Apply** or **Save**.

You can also use the -customProperties flag in the setAppActiveSecuritySettings wsadmin command to modify an existing custom property in a global security domain configuration. See the SecurityConfigurationCommands command group for the AdminTask object article for more information about this command. Use the -securityDomainName flag to specify the security domain where the custom property is located. For example:

```
wsadmin>AdminTask.setAppActiveSecuritySettings('[ -securityDomainName
testDomain -customProperties ["com.ibm.websphere.security.test=false"]]'
```

Deleting an existing custom property in a global security configuration or in a security domain configuration

Custom properties are arbitrary name-value pairs of data, where the name is a property key and the value is a string value that can be used to set internal system configuration properties. Defining a new property enables you to configure settings beyond those that are available in the administrative console. You can delete existing security custom properties in a global security configuration or in a security domain configuration.

About this task

Deleting an existing custom property in a global security configuration using the administrative console

1. Click **Security > Global security > Custom properties**.
2. Select the custom property you want to delete.
3. Click **Delete**.
4. Click **Apply** or **Save**.

You can also use the -customProperties flag in the setAdminActiveSecuritySettings wsadmin command to delete an existing custom property in a global security configuration. See the SecurityConfigurationCommands command group for the AdminTask object article for more information about this command. For example:

```
wsadmin>AdminTask.setAdminActiveSecuritySettings('[-customProperties
["com.ibm.websphere.security.test="]]')
```

Deleting an existing custom property in a security domain configuration using the administrative console

1. Click **Security > Security domains**.
2. Click **Custom properties**.
3. Select the custom property you want to delete.
4. Click **Delete**.
5. In the **Value** field, enter the value you want to delete.
6. Click **Apply** or **Save**.

You can also use the -customProperties flag in the setAppActiveSecuritySettings wsadmin command to delete an existing custom property in a global security domain configuration. See the SecurityConfigurationCommands command group for the AdminTask object article for more information about this command. Use the -securityDomainName flag to specify the security domain where the custom property is located. .For example:

```
wsadmin>AdminTask.setAppActiveSecuritySettings('[ -securityDomainName testDomain
-customProperties ["com.ibm.websphere.security.test="]]')
```

Chapter 6. Configuring multiple security domains

By default, all administrative and user applications in WebSphere Application Server use the global security configuration. For example, a user registry defined in global security is used to authenticate users for every application in the cell. Out-of-the-box, this behavior is the same as it was in previous releases of WebSphere Application Server. You can create additional WebSphere security domains if you want to specify different security attributes for some or all of your user applications. This section describes how to configure a security domain by using the administrative console.

Before you begin

Only users assigned to the administrator role can configure or create new multiple security domains. Enable global security in your environment before configuring multiple security domains.

Read about “Multiple security domains” on page 126 for a better understanding of what multiple security domains are and how they are supported in this version of WebSphere Application Server.

About this task

Security domains enable you to define multiple security configurations for use in your environment. For example, you can define different security (such as a different user registry) for user applications than for administrative applications. You can also define separate security configurations for user applications deployed to different servers and clusters.

Perform the following steps to configure a new security domain by using the administrative console:

Procedure

1. Click **Security > Security domains**.
2. If you are creating a new multiple security domain, click **New**. Supply a unique name and description for the domain and click **Apply**. If you want to configure an existing multiple security domain, select one to edit. Once you click **Apply** the domain name and additional sections are displayed. One section enables you to define the security attributes for the domain, and another section enables you to select the scopes to which the domain applies.
3. Under Assigned Scopes, select whether you want to assign the security domain to the entire cell or if you want to select the specific servers, clusters, and service integration buses to be included in the security domain. The Assigned Scopes section has two views. The default view is a cell topology. To assign the security domain to the entire cell, click the check box for the cell and then click **Apply** or **OK**.

   The name of the security domain appears next to the cell name, which indicates that the domain is now assigned to the cell. You can expand the topology and assign the domain to one or more servers and clusters. When an item in the topology is already assigned to another security domain, the check box is disabled and the name of the assigned domain is displayed to the right of the scope name. If you want to assign one of these scopes to the domain, you must first disassociate it with its current domain.

   Select **All assigned scopes** to view a list of only those resources that are currently assigned to the security domain.
4. Customize your security configuration by specifying security attributes for your new domain. Attributes that are not listed can not be customized at the domain level. Domains inherit attributes from the global security configuration.

   There are twelve individually configurable security attribute sections. You can expand and collapse each section. In the collapsed state, the name and a summary value for the section are displayed.

Additionally, the summary value text indicates whether the attribute is defined in global security and is reused by the domain (as indicated by gray text) or if it is customized for the domain (as indicated by black text prefixed by the word “Customized”).

Initially, each security attribute is set to use the global security settings. When an attribute is set to use global security, there is no domain-specific configuration for that attribute. Applications that use the domain use the global configuration for these security attributes.

Only configure the security attributes that you want to change. To configure a security attribute for a domain, expand the security attribute section. The key properties of the global configuration display beneath the **Use global security** option. These properties are provided for convenience.

To customize the configuration for the domain, select **Customize for this domain**. Configure the property and then click **OK** or **Apply**.

Note: In general, when you select **Customize for this domain**, you override all of the security configurations that are defined for that section in global security. Application logins, system logins, and J2C authentication data entries are some exceptions. When you define entries for a domain, applications in the domain are able to access the global entries in addition to the domain-specific entries.

For example, you might want to use a different user registry for applications that use the security domain but also want to use the global security configuration for all of the other security properties. In this case, expand the User Realm section and select **Customize for this domain**. Select a user registry type, click **Configure**, and provide the appropriate configuration details on the subsequent panel.

You can change security attributes such as the following:

Application Security

Specifies the settings for application security and Java 2 security. You can use the global security settings or customize the settings for a domain.

Select **Enable application security** to enable or disable security this choice for user applications. When this selection is disabled, all of the EJBs and web applications in the security domain are no longer protected. Access is granted to these resources without user authentication. When you enable this selection, the J2EE security is enforced for all of the EJBs and web applications in the security domain. The J2EE security is only enforced when Global Security is enabled in the global security configuration, (that is, you cannot enable application security without first enabling Global Security at the global level).

Java 2 Security

Select **Java 2 security** to enable or disable Java 2 security at the domain level. This choice enables or disables Java 2 security at the process (JVM) level so that all applications (both administrative and user) can enable or disable Java 2 security.

User realm

This section enables you to configure the user registry for the security domain. You can separately configure any registry that is used at the domain level. Read about “Multiple security domains” on page 126 for more information.

Trust association

When you configure the trust association interceptor (TAI) at a domain level, the interceptors configured at the global level are copied to the domain level for convenience. You can modify the interceptor list at the domain level to fit your needs. Only configure those interceptors that are to be used at the domain level.

SPNEGO Web Authentication

The SPNEGO web authentication, which enables you to configure SPNEGO for web resource authentication, can be configured at the domain level.

Note: In WebSphere Application Server Version 6.1, a TAI that uses the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) to securely negotiate and authenticate HTTP requests for secured resources was introduced. This function was deprecated in WebSphere Application Server 7.0. SPNEGO web authentication has taken its place to provide dynamic reload of the SPNEGO filters and to enable fallback to the application login method.

RMI/IIOP Security

The RMI/IIOP security attribute refers to the CSIv2 (Common Secure Interoperability version 2) protocol properties. When you configure these attributes at the domain level, the RMI/IIOP security configuration at the global level is copied for convenience.

You can change the attributes that need to be different at the domain level. The Transport layer settings for CSIv2 inbound communications should be the same for both the global and the domain levels. If they are different, the domain level attributes are applied to all of the application in the process.

JAAS application logins

Specifies the configuration settings for the Java Authentication and Authorization Service (JAAS) application logins. You can use the global security settings or customize the settings for a domain.

Note: The JAAS application logins, the JAAS system logins, and the JAAS J2C authentication data aliases can all be configured at the domain level. By default, all of the applications in the system have access to the JAAS logins configured at the global level. The security runtime first checks for the JAAS logins at the domain level. If it does not find them, it then checks for them in the global security configuration. Configure any of these JAAS logins at a domain only when you need to specify a login that is used exclusively by the applications in the security domain.

JAAS system logins

Specifies the configuration settings for the JAAS system logins. You can use the global security settings or customize the configuration settings for a domain.

JAAS J2C authentication

Specifies the configuration settings for the JAAS J2C authentication data. You can use the global security settings or customize the settings for a domain.

Java Authentication SPI (JASPI)

Specifies the configuration settings for a Java Authentication SPI (JASPI) authentication provider and associated authentication modules. You can use the global security settings or customize the settings for a domain. To configure JASPI authentication providers for a domain, select **Customize for this domain** and then enable JASPI. Select **Providers** to define providers for the domain.

Note: The JASPI authentication provider can be enabled with providers configured at the domain level. By default, all of the applications in the system have access to the JASPI authentication providers configured at the global level. The security runtime first checks for the JASPI authentication providers at the domain level. If it does not find them, it then checks for them in the global security configuration. Configure JASPI authentication providers at a domain only when the provider is to be used exclusively by the applications in that security domain.

Authentication Mechanism Attributes

Specifies the various cache settings that need to applied at the domain level.

Select **Authentication cache settings** to specify your authentication cache settings. The configuration specified on this panel is applied only to this domain.

Select **LTPA Timeout** to configure a different LTPA timeout value at the domain level. The default timeout value is 120 minutes, which is set at the global level. If the LTPA timeout is set at the domain level, any token that is created in the security domain when accessing user applications is created with this expiration time.

When **Use realm-qualified user names** is enabled, user names returned by methods such as `getUserPrincipal( )` are qualified with the security realm (user registry) used by applications in the security domain.

Authorization Provider

You can configure an external third party JACC (Java Authorization Contract for Containers) provider at the domain level. Tivoli Access Manager's JACC provider can only be configured at the global level. Security domains can still use it if they do not override the authorization provider with another JACC provider or with the built-in native authorization.

Custom properties

Set custom properties at the domain level that are either new or different from those at the global level. By default, all of the custom properties at the global security configuration can be accessed by all of the applications in the cell. The security runtime code first checks for the custom property at the domain level. If it does not find it, it then attempts to obtain the custom property from the global security configuration.

5. Once you have configured the security attributes and assigned the domain to one or more scopes, click **Apply** or **OK**.
6. Restart all servers and clusters for your changes to take effect.

Multiple security domains

The WebSphere Security Domains (WSD) provide the flexibility to use different security configurations in WebSphere Application Server. The WSD is also referred to as multiple security domains, or simply, security domains. You can configure different security attributes, such as the UserRegistry, for different applications.

Note: Multiple security domain support was introduced in WebSphere Application Server Version 7.0. You can create different security configurations and assign them to different applications in WebSphere Application Server processes. By creating multiple security domains, you can configure different security attributes for both administrative and user applications within a cell environment. You can configure different applications to use different security configurations by assigning the servers or clusters or service integration buses that host these applications to the security domains. Only users assigned to the administrator role can configure multiple security domains.

The following sections describe multiple security domains in more detail:

- “Security domains in a mixed-version environment” on page 141
- “Modifying security domains” on page 142

Why security domains are useful

WebSphere Security Domains provide two major benefits:

- WebSphere Application Server administrative applications can be configured with a different set of security configurations from those for user applications.
- They enable one set of applications to have a different set of security configurations from another set of applications.

  For example, WebSphere Application Server administration can be configured to a user registry of federated repository while the applications can be configured to a user registry of LDAP.

In previous versions of WebSphere Application Server, all administrative and user applications use security attributes different from those attributes that are defined in global security. All administrative and user applications in WebSphere Application Server use global security attributes by default. For example, a user registry defined in global security is used to authenticate a user for every application in the cell.

In this release of WebSphere Application Server, however, you can use multiple security attributes for user applications other than the global security attributes, create a security domain for those security attributes that must differ, and associate them with the servers and clusters that host those user applications. You also can associate a security domain with the cell. All of the user applications in the cell use this security domain if they do not have a domain previously associated with them. However, global security attributes are still required for administrative applications such as the administrative console, naming resources and MBeans.

If you have used server level security in previous releases of WebSphere Application Server, you should now use multiple security domains since they are more flexible and easier to configure.

Server level security is deprecated in this release. Read “Relationship between global security and security domains” for more information.

Relationship between global security and security domains

Global Security applies to all administrative functions and the default security configuration for user applications. Security domains can be used to define a customized configuration for user applications.

You must have a global security configuration defined before you can create security domains. The global security configuration is used by all of the administrative applications such as the administrative console, naming resources, and Mbeans. If no security domains are configured, all of the applications use information from the global security configuration. User applications such as Enterprise JavaBeans (EJBs), servlets and administrative applications use the same security configuration.

When you create a security domain and associate it with a scope, only the user applications in that scope use the security attributes that are defined in the security domain. The administrative applications as well as the naming operations in that scope use the global security configuration. Define the security attributes at the domain level that need to be different from those at the global level. If the information is common, the security domain does not need to have the information duplicated in it. Any attributes that are missing in the domain are obtained from the global configuration. The global security configuration data is stored in the `security.xml` file, which is located in the `$WAS_HOME/profiles/$ProfileName/cells/$CellName` directory.

The following figure provides an example of a security multiple domain where the cell, a server and a cluster are associated with different security domains. As shown in the figure, the user applications in server **S1.1** as well as the cluster use security attributes that are defined in **Domain2** and **Domain3**

respectively (since these scopes are associated with these domains). Server **S2.2** is not associated with a domain. As a result, the user application in **S2.2** uses the domain that is associated with the cell (**Domain1**) by default . Security attributes that are missing from the domain level are obtained from the global configuration.

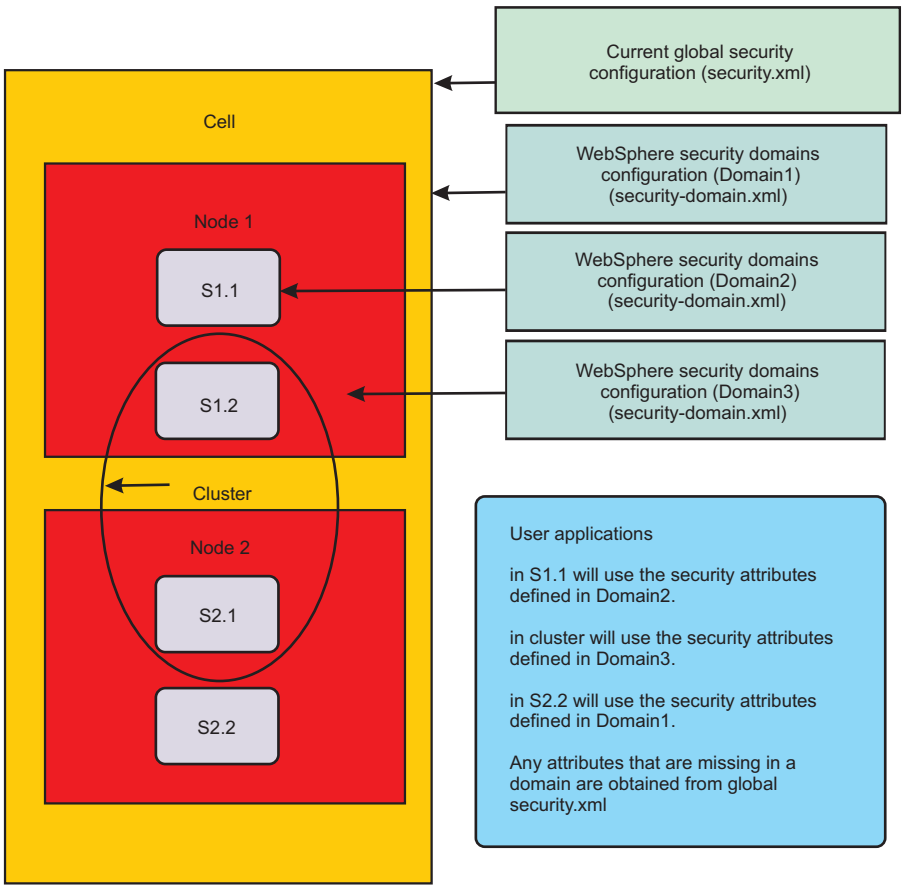


Figure 1. Scopes that can be associated to a security domain

The following figure shows the various high-level security attributes that can be defined at the global security configuration and those that can be overridden at the domain level.

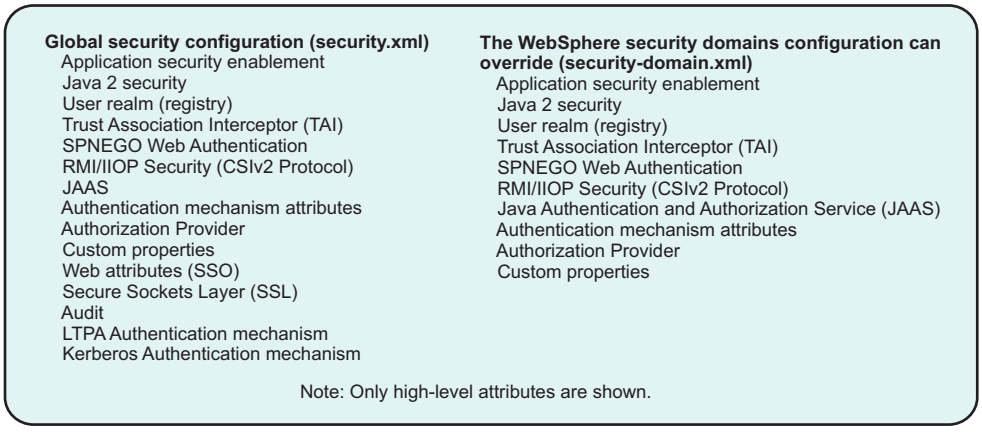


Figure 2. Security attributes that can be configured at the security domain

Contents of a security domain

A security domain is represented by two configuration files. One configuration file contains the list of attributes that are configured in the security domain. The other configuration file contains the scopes that use the security domain. The security domain information is stored in the `$WAS_HOME/profiles/$ProfileName/config/waspolicies/default/securitydomains/$SecurityDomainName` directory. For every security domain that is configured, a `$SecurityDomainName` directory is created with two files in it: the `security-domain.xml` file contains the list of security attributes configured for the security domain, and the `security-domain-map.xml` file contains the scopes that use the security domain.

The following figure indicates the location of the main security domain related files and the contents of those files.

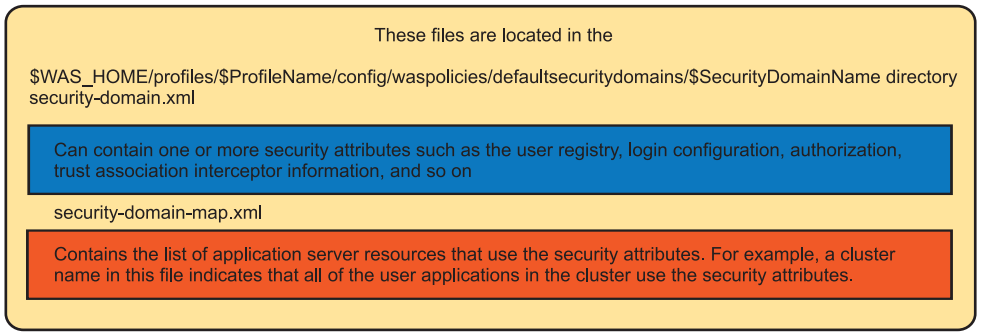


Figure 3. Location and contents of the main security domain related files

Note: You should not modify these files manually. Use administrative console tasks or scripting commands to modify the files instead. For a complete list of administrative tasks and scripting commands, see the links in "Related tasks" at the bottom of this document.

Creating security domains

Use the administrative console tasks or scripting commands to create security domains. In the administrative console, access security domains by clicking **Security** > **Security domains**. Help is available for each administrative console panel.

For a complete list of administrative console tasks and scripting commands, see the links in "Related tasks" at the bottom of this document.

When you create a security domain you must supply a unique name for the domain, the security attributes you want to configure for the security domain, and the scopes that need to use the security domain. Once configured, the servers that use the security domain must be restarted. The user applications in those scopes then use the attributes that are defined in the security domain. Any attributes that are not configured at the domain level are obtained from the global security configuration. Administrative applications and naming operations in the scopes always use the security attributes from the global security configuration. You must actively manage these attributes.

Any new security domain attributes must be compatible with those global security attributes that are inherited by the user applications that are assigned to the domain.

Other than for JAAS and custom properties, once global attributes are customized for a domain they are no longer used by user applications.

The security domains panel in the administrative console enables you to assign resources and to select the appropriate security attributes for your domain. The panel displays the key security attributes at the global configuration; you can make the decision to override them at the domain level if necessary. Once you have configured and saved the attributes at the domain level, the summary value on the panel displays the customized value for the domain (tagged with the word "customized" in black text).

A scope (a server, cluster, service integration bus or a cell) can be associated with only one domain. For example, you cannot define two domains that both have the cell-wide scope. Multiple scopes, however, can be defined in the same security domain. For example, a domain can be scoped to **Server1** and to **Server2** only within the cell.

The assigned scopes section on the security domain panel displays two views: one view that enables you to select and assign scopes to the domain, and another view that enables you to see a list of the currently assigned scopes. For convenience, you also have the flexibility to copy all of the security attributes from an existing security domain or the global configuration into a new security domain, and then modify only those attributes that must be different. You must still associate the scopes to these copied domains.

Scripting commands also provide you with the ability to create, copy and modify security domains. Once you create a domain, you must run the appropriate commands to associate security attributes and scopes to it.

Configuring attributes for security domains

Security attributes that can be configured at the domain level in WebSphere Application Server Version 8.5 are:

- Application security
- Java 2 security
- User realm (registry)
- Trust association
- Simple and Protected GSS-API Negotiation (SPNEGO) web authentication
- RMI/IIOP security (CSIv2)
- JAAS logins (Application, System and J2C Authentication Data)
- Java Authentication SPI
- Authentication mechanism attributes
- Authorization provider
- Federated repositories
- Custom properties

The security domains panels in the administrative console display all of these security attributes.

Some of the other well-known attributes that you cannot override at the domain level are Kerberos, Audit, Web Single Sign-on (SSO) and Tivoli Access Manager (TAM). The Secure Socket Layer (SSL) attribute already supports different scopes, but it is not part of the domain configuration. For all of the attributes that are not supported at the domain level, user applications in a domain share their configuration from the global level.

Any new security domain attributes must be compatible with those global security attributes that are inherited by the user applications that are assigned to the domain. You must actively manage these attributes. For example, if you customize only a JAAS configuration at the domain level you must make sure that it works with the user registry configured at the global level (if the user registry is not customized at the domain level).

Other than for JAAS and custom properties, once global attributes are customized for a domain they are no longer used by user applications.

The Tivoli Access Manager client runtime is used to provide authentication (used by TrustAssociationInterceptor and PDLoginModule) and authorization (used for JACC) by contacting TAM servers. There is only one Tivoli Access Manager runtime shared by all servers in a cell. Read the Tivoli Access Manager JACC provider configuration topic for more information.

You cannot have a different Tivoli Access Manager configuration at the security domain level to override the configuration at the cell level. However, you can to some degree specify Trust Association Interceptor (TAI) and JACC configuration at the security domain level. For example, you can use a different TAI or a different authorization provider. Since TAM server connectivity can only be defined at the global level, you can have a variety of TAIs defined and configured at the security domain level. Some of these TAIs might not use the TAM user repository, while others do. The TAIs that do need to connect to TAM will also connect to the globally-defined TAM server. Similarly, for authorization, you can have a variety of external authorization providers configured at the domain level. However, if any of these external authorization providers require connection to TAM they end up talking to the singular globally-configured TAM server.

Associating scopes to security domains

In WebSphere Application Server Version 8.5, you can associate a security domain at the cell level, the server level, the cluster level and the service integration bus level.

Note: For more information about the service integration bus and bus security in multiple security domains for WebSphere Application Server Version 8.5, see Messaging security and multiple security domains.

When a security domain is associated with a server that is not part of a cluster, all user applications in that server use the attributes from the security domain. Any missing security attributes are obtained from the global security configuration. If the server is part of a cluster, you can associate the security domain with the cluster but not with the individual members in that cluster. The security behavior then remains consistent across all of the cluster members.

If a server is to be part of a cluster, create a cluster first and associate the security domain to it. You might have associated a domain to a server before it was a member of a cluster. If so, even though the domain is associated with the server directly, the security runtime code does not look at the domain. When a server is a cluster member, the security runtime disregards any security domains associated directly to the server. Remove the server scope from the security domain and associate the cluster scope to it instead.

A security domain can also be associated to the cell. This is usually done when you want to associate all user applications in WebSphere Application Server to a security domain. In this scenario, all of the administrative applications and the naming operations use the global security configuration while all of the user applications use the domain level configuration. If you want to split the security configuration information for administrative and user applications, this is all that is needed.

If you have a mixed-version environment, or plan to have one in future, and you want to associate security domains at the cell level, read “Security domains in a mixed-version environment” on page 141 for more information.

If you are on a base profile server that has its own security domain defined, which is then federated to a deployment manager, associate the server scope to the security domain and not the cell scope. When you federate that node, the security domain information is propagated to the deployment manager. If the cell scope is associated to it, the network deployment configuration uses this security configuration, which might impact existing applications. During federation, the cell scope is changed to the server scope that is being federated. If the server scope is associated with the security domain, only that server uses the security domain after the federation. Other applications in other servers and clusters are not impacted. However, if this base profile server is registered to the Administrative Agent process you can associate the cell scope to the security domain if you want all of the servers from the base profile to use the same security domain for all of their user applications. Read about “Federating a node with security domains” on page 141 for more information.

You can have a security domain associated at the cell level and also other security domains associated to various clusters or individual servers (those that are not part of any clusters). In this case, the security runtime first checks if any security domains are associated with the server or a cluster. If there is a security domain associated with the server or a cluster, the security attributes defined in it are used for all of the applications in that server or cluster. Any security attributes missing from this server or cluster domain are obtained from the global security configuration, and not from the domain configuration associated with the cell.

If the server or cluster does not have its own domain defined, the security runtime code uses the security attributes from the domain associated with the cell (if one is defined). Any security attributes missing from the cell domain are inherited from the global security configuration.

Relationship between old server level security and the new security domains

In previous releases of WebSphere Application Server, you could associate a small set of security attributes at a server level. These attributes were used by all of the applications at the server level. The previous way of configuring the security attributes was deprecated in WebSphere Application Server 7.0, and will be removed in a future release.

You should now use the new security domains support starting in WebSphere Application Server 7.0, as these security domains are more easily managed and much more flexible. For example, in previous versions of WebSphere Application Server, you must manually associate the same security configuration to all of the cluster members by configuring the same security attributes for every server in a cluster.

The migration tool migrates the existing server level security configuration information to the new security domain configuration when the script compatibility mode is `false` (`-scriptCompatibility="false"`). A new security domain is created for every server security configuration if it is not part of a cluster. If it is part of a cluster, a security domain is associated with the cluster instead of with all of the servers in that cluster. In both cases, all of the security attributes that were configured at the server level in previous releases are migrated to the new security domain configuration, and the appropriate scope is assigned to the security domains.

If the script compatibility mode is set to `true`, the server level security configuration is not migrated to the new security domains configuration. The old server security configuration is migrated without any changes. The security runtime detects that the old security configuration exists and uses that information, even if a security domain is associated either directly or indirectly to the server. If the script compatibility mode is set to `true`, remove the security configuration from the server level and then create a security domain with the same set of security attributes.

How domain level security attributes are used by security runtime and applications

This section describes how the individual attributes at the domain level are used by the security runtime and how that impacts the user application security. Since all of these security attributes are also defined at the global level, more information about these attributes can be obtained elsewhere. For the purposes of this section, the emphasis is on domain level behavior.

1. Application Security:

   Select Enable application security to enable or disable security for user applications. When this selection is disabled, all of the EJBs and web applications in the security domain are no longer protected. Access is granted to these resources without user authentication. When you enable this selection, the J2EE security is enforced for all of the EJBs and web applications in the security domain. The J2EE security is only enforced when Global Security is enabled in the global security configuration, (that is, you cannot enable application security without first enabling Global Security at the global level).

2. Java 2 Security:

   Select Use Java 2 security to enable or disable Java 2 security at the domain level or to assign or add properties related to Java 2 security. This choice enables or disables Java 2 security at the process (JVM) level so that all applications (both administrative and user) can enable or disable Java 2 security.

3. User Realm (User Registry):

   This section enables you to configure the user registry for the security domain. You can separately configure any registry that is used at the domain level. Read about “Configuring attributes for security domains” on page 130 for more information.

   When configuring a registry at the domain level you can choose to define your own realm name for the registry. The realm name distinguishes one user registry from another. The realm name is used in multiple places – in the Java client login panel to prompt the user, in the authentication cache, and when using native authorization.

   At the global configuration level, the system creates the realm for the user registry. In previous releases of WebSphere Application Server, only one user registry is configured in the system. When you have multiple security domains you can configure multiple registries in the system. For the realms to be unique in these domains, configure your own realm name for a security domain. You also can choose the system to create a unique realm name if it is certain to be unique. In the latter case, the realm name is based on the registry that is being used.

   For LDAP registries, the host:port of the LDAP server is the system-generated realm name. For localOS, the name of the localOS machine is the realm name. For custom user registries, the realm is the one returned by the `getRealm ( )` method of the custom registry implementation.

   If the system generated realm names are unique enough, you can choose the option for the system to generate the realm name. If not, choose a unique realm name for each security domain where you have the user registry configured. If the underlying user repository is the same, use the same realm name in different domains. From a security runtime perspective, same realm names have the same set of users and groups information. For example, when users and groups information is required from a realm, the first user repository that matches the realm is used.If a localOS registry that is not centralized is configured for any domain, and that domain is associated with servers or clusters in nodes not on the same system as the deployment manager, the realm name has to be provided. This realm name has to be the same as it would be if it were generated on the node. This realm name can be obtained by calling the `getRealm()` method on the SecurityAdmin MBean on that node. Typically, the realm name for localOS registries is the hostname of the machine. In this case, you should not let the system generate the realm name but rather get the realm name that is used by the processes in the node.

   If you select the system to generate the realm for the localOS registry at the time of the user registry configuration, it chooses the localOS registry that is used by the deployment manager. If the realm configured does not match the realm used by the servers then there are authorization issues. Also

note that in this case, the domain using this local registry can only be associated with servers and clusters that belong to nodes on the same machine.

In WebSphere Application Server Version 7.0, the federated repositories user registry can only be configured at the global level and have only one instance per cell, but any domain can use it by configuring it as the active registry. In WebSphere Application Server Version 8.0, you can configure a unique instance of a federated repository at the domain level in a multiple security domain environment.

When a security domain is copied from the global level, the users and groups defined at the global level are also copied to the security domain. This is also true when copying from an existing domain. A newly-created security domain that uses the file-based VMM repository requires that the user populate the repository with users and groups.

Also new in this release of WebSphere Application Server, a new checkbox on the Realm configurations settings administrative console page, Use global schema for model, sets the global schema option for the data model in a multiple security domain environment. Global schema refers to the schema of the admin domain.

When more than one user registry is in a process, the naming lookup that uses “UserRegistry” as the lookup name returns the user registry that is used by user applications. The user registry used by administrative applications is bound by the lookup name, “AdminUserRegistry”.

As described in “Cross realm communication” on page 138, when an application in one realm communicates with an application in another realm using LTPA tokens, the realms have to be trusted. The trust relationship can be established using the **Trusted authentication realms** – inbound link in the user registry panel or by using the **addTrustedRealms** command. You can establish trust between different realms. A user logged into one realm can access resources in another realm. If no trust is established between the two realms the LTPA token validation fails.

Note: The realm name used in the web.xml file is not related to the user registry realm.

4. Trust Association:

   When you configure the trust association interceptor (TAI) at a domain level, the interceptors configured at the global level are copied to the domain level for convenience. You can modify the interceptor list at the domain level to fit your needs. Only configure those interceptors that are to be used at the domain level.

   Tivoli Access Manager's trust association interceptors can only be configured at the global level. The domain configuration can also use them, but cannot have a different version of the trust association interceptor. Only one instance of Tivoli Access Manager's trust association interceptors can exist in the cell.

5. SPNEGO web authentication:

   The SPNEGO web authentication, which enables you to configure SPNEGO for web resource authentication, can be configured at the domain level.

   Note: In WebSphere Application Server Version 6.1, a TAI that uses the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) to securely negotiate and authenticate HTTP requests for secured resources was introduced. In WebSphere Application Server 7.0, this function was deprecated. SPNEGO web authentication has taken its place to provide dynamic reload of the SPNEGO filters and to enable fallback to the application login method.

6. RMI/IIOP Security (CSIv2):

   The RMI/IIOP security attribute refers to the CSIv2 (Common Secure Interoperability version 2) protocol properties. When you configure these attributes at the domain level, the RMI/IIOP security configuration at the global level is copied for convenience.

   You can change the attributes that need to be different at the domain level. The Transport layer settings for CSIv2 inbound communications should be the same for both the global and the domain levels. If they are different, the domain level attributes are applied to all of the application in the process.

When a process communicates with another process with a different realm, the LTPA authentication and the propagation tokens are not propagated to the downstream server unless that server is listed in the outbound trusted realms list. This can be done using the **Trusted authentication realms** – outbound link on the **CSIv2** outbound communication panel, or by using the **addTrustedRealms** command task. Read about “Cross realm communication” on page 138 for more information.

7. JAAS (Java Authentication and Authorization Service):

   The JAAS application logins, the JAAS system logins, and the JAAS J2C authentication data aliases can all be configured at the domain level. By default, all of the applications in the system have access to the JAAS logins configured at the global level. The security runtime first checks for the JAAS logins at the domain level. If it does not find them, it then checks for them in the global security configuration. Configure any of these JAAS logins at a domain only when you need to specify a login that is used exclusively by the applications in the security domain.

   For JAAS and custom properties only, once global attributes are customized for a domain they can still be used by user applications.

8. Java Authentication SPI (JASPI)

   Specifies the configuration settings for a Java Authentication SPI (JASPI) authentication provider and associated authentication modules to be applied at the domain level.

   Select **Providers** to create or to edit a JASPI authentication provider.

   Note: The JASPI authentication provider can be enabled with providers configured at the domain level. By default, all of the applications in the system have access to the JASPI authentication providers configured at the global level. The security runtime first checks for the JASPI authentication providers at the domain level. If it does not find them, it then checks for them in the global security configuration. Configure JASPI authentication providers at a domain only when the provider is to be used exclusively by the applications in that security domain.

9. Authentication Mechanism Attributes:

   Specifies the various cache settings that must be applied at the domain level.

   a. Authentication cache settings - use to specify your authentication cache settings. The configuration specified on this panel is applied only to this domain.

   b. LTPA Timeout - You can configure a different LTPA timeout value at the domain level. The default timeout value is 120 minutes, which is set at the global level. If the LTPA timeout is set at the domain level, any token that is created in the security domain when accessing user applications is created with this expiration time.

   c. Use realm-qualified user names - When this selection is enabled, user names returned by methods such as **getUserPrincipal( )** are qualified with the security realm (user registry) used by applications in the security domain.

10. Authorization Provider:

    You can configure an external third party JACC (Java Authorization Contract for Containers) provider at the domain level. Tivoli Access Manager's JACC provider can only be configured at the global level. Security domains can still use it if they do not override the authorization provider with another JACC provider.

    The JACC attributes, for example the Policy object, are based at the JVM level. This implies that there can be only be one JACC policy object in a JVM process. However, when you have multiple JACC providers configured, the deployment manager process has to handle all these providers in the same JVM because it has to propagate the authorization policy of applications to the respective provider based on the application name.

    If your JACC provider can handle propagating the authorization policy to multiple providers, you can configure it at the global level. In this case, when an application is installed, this JACC provider is called in the deployment manager process and it is the responsibility of this JACC provider to propagate the information to the corresponding JACC provider based on the application name passed in the contextID.

Another way to achieve this is to set the custom property, `com.ibm.websphere.security.allowMultipleJaccProviders=true`, at the global security level. When this property is set, WebSphere Application Server propagates the authorization policy information to the JACC provider associated with the domain that corresponds to the target server where the application is installed. This property is only used at the deployment manager process since the managed servers do not host multiple JACC providers.

11. Custom properties:

    Set custom properties at the domain level that are either new or different from those at the global level. By default, all of the custom properties at the global security configuration can be accessed by all of the applications in the cell. The security runtime code first checks for the custom property at the domain level. If it does not find it, it then attempts to obtain the custom property from the global security configuration.

    For JAAS and custom properties only, once global attributes are customized for a domain they can still be used by user applications.

Client and application security programming model when using security domains

A Java client or an application acting as a client that accesses an EJB typically does a naming lookup first. The naming resource, which is used by both administrative and the user applications, is considered an administrative resource. It is protected by the global security configuration information. In a multiple domain setup where the global security is using one realm (the user registry) and a domain is using a different realm, the Java client must authenticate to two different realms. The first authentication is required for the realm in the global security configuration for the naming operation to succeed, and the second authentication is required to access the EJB, which uses a different realm.

The **CosNamingRead** role protects all naming read operations. This role is usually assigned the **Everyone** special subject. This implies that any user, valid or not, can look up the name space. When a multiple domain is defined, if the **CosNamingRead** role has the **Everyone** special subject the security runtime code in the client side does not prompt you to log in. It uses the UNAUTHENTICATED subject to access the naming operation instead. Once the naming lookup operation is completed, when the client attempts to access the EJB it is prompted with a login panel that indicates the realm that is currently used by that EJB application (that is, the realm used in the domain). The client then presents the appropriate user credentials for that realm, which can then access the EJB. This logic applies to all variations of login source, including **properties** and **stdin**, not just when the login source is set to **prompt**.

If the **Everyone** special subject is removed from the **CosNamingRead** role, you are prompted twice. If the login source is **properties**, you can uncomment the **com.ibm.CORBA.loginRealm** property in the `$WAS_HOME/profiles/$ProfileName/properties/sas.client.props` file and add the appropriate realms using “|” as the separator. You must also enter the corresponding users and passwords in the **com.ibm.CORBA.loginUserid** and **com.ibm.CORBA.loginPassword** properties respectively. When you are using the programmatic logon in the Java client code you must authenticate twice with different user credentials; once prior to do a naming lookup for the EJB (the user should be in the global realm), and later prior to calling any method in the EJB (the user should be in the EJB domain's realm).

In general, when a Java client needs to authenticate to multiple and different realms it has to provide the credential information for all of those realms. If the login source is **prompt** or **stdin** it is prompted to login multiple times, once for each realm. If the login source is set to **properties**, the appropriate properties in the `sas.client.props` file (or any related file) are used for authenticating to different realms.

In certain scenarios, a client might make multiple calls to the same realm. For example, the Java client can access a resource using **realm1** followed by access to a resource using **realm2**, and then come back to access a resource in **realm1** again. In this case, the client is prompted three times; first for **realm1**, secondly for **realm2** and finally for **realm1** again.

By default, the subject that is used to login at a realm is not cached by the client side code. If you have this scenario, and you want the client to cache the subject based on the realm, set the **com.ibm.CSI.isRealmSubjectLookupEnabled** property to `true` in the `sas.client.props` file. If the **com.ibm.CSI.isRealmSubjectLookupEnabled** property is set, the client code caches the subject based on the realm name. The next time the Java client needs to authenticate to this realm, the cache is located to obtain the subject and the client is not prompted. Also, when the **com.ibm.CSI.isRealmSubjectLookupEnabled** property is set, the same subject that was logged in the first time is used for subsequent logins. If the subject information needs to change then this property should not be set.

If the client is doing a programmatic login it can pass the realm along with the user and password that it needs to authenticate. In this case, when the **com.ibm.CORBA.validateBasicAuth** property is set to `true` (the default value) in the `sas.client.props` file, the registry that matches the realm name is used for login. That realm must be supported in the process where the authentication takes place.

When using the WSLogin JAAS configurations, you also must set the `use_realm_callback` option in the `wsjaas_client.config` file in `$WAS_HOME/profiles/$ProfileName/properties` for the realm name to be passed to the call back handler. If you want to specify a different provider URL for the name server, set the `use_appcontext_callback` option and pass in the provider URL properties in a hash map to WSLogin.

If you do not know the realm name, use `<default>` as the realm name. The authentication is performed against the application realm. If the naming read operation does not have the `Everyone` special subject assigned, you must provide the realm that is used by the administrative applications (the registry used in the global security configuration), as well as the appropriate user and password information in that registry for the lookup operation to succeed.

After the lookup operation succeeds, perform another programmatic login by providing the application realm (or `<default>`) and the user and password information for the appropriate user in the registry that is used by the application. This is similar to the case where the login source is `prompt`. You must authenticate twice, once for the registry used by the global security configuration (for the naming lookup operation) and again for the registry used by the application to access the EJB.

If **com.ibm.CORBA.validateBasicAuth** is set to `false` in the `$WAS_HOME/profiles/$ProfileName/properties/sas.client.props` file then the programmatic login can use `<default>` as the realm name for both the lookup and the EJB operations. The actual authentication occurs only when the resource is accessed on the server side, in which case the realm is calculated based on the resource that is accessed.

The new security domain support starting in WebSphere Application Version 7.0 does not change the current application security programming model. However, it provides more flexibility and capabilities such as the following:

- User applications can still find the user registry object by using the naming lookup for “UserRegistry”. For the registry object used by administrative applications, the naming lookup for “AdminUserRegistry” can be used.
- The application usage of the JAAS login configuration does not change in a multiple domain setup. However, if an application must refer to the JAAS configuration that is specified at the domain level, the administrator and the deployer of that application must make sure that this domain is configured with the JAAS configurations that are required by the application.
- If an application needs to communicate with other applications using different realms, trust relationship should be established for both inbound and outbound communications when using the LTPA tokens. Read about “Cross realm communication” on page 138 for more information.
- When using programmatic login in the applications, if you want to login to the realm used by the application, use `<default>` as the realm name or provide the realm name that the application is using. If you need to login to the global realm, you must provide the global realm name. If you provide any other

realm, only a basic authentication subject is created. When the request actually flows to the server hosting that realm, the actual authentication of the user occurs if that server hosts the realm. If the server does not host the realm, the login fails.

Application deployment in multiple domains configurations

When deploying an application in a multiple domain setup, all of the modules in the application should be installed in the servers or clusters that belong to the same security domain. If not, depending on the security attributes configured in these security domains, inconsistent behavior can result. For example, if the domains contain different user registries, the users and groups information can be different, which can cause inconsistent behavior when accessing the modules. Another example is when the JAAS data is different between the security domains. The JAAS configurations is not accessible from all of the modules in the application. The security runtime code and the command tasks rely on one domain being associated with an application when dealing with attributes such as user registry, JAAS login configurations, J2C authentication data, and authorization.

In most cases, application deployment fails when an application is deployed across different domains. However, since this was possible in earlier releases of WebSphere Application Server when only a few attributes were supported at the server level, the deployment tool first checks for attributes that are configured at the domains. If the attributes in the domain are the same as those supported in previous releases, the administrative console requests confirmation to ensure that you want to deploy application modules across multiple security domains. Unless there is an absolute requirement to deploy the applications across different domains, stop the deployment and select the servers and clusters in the same security domain.

Cross realm communication

When applications communicate using the RMI/IIOP protocol and LTPA is the authentication mechanism, the LTPA token is passed between the servers involved. The LTPA token contains the realm-qualified uniqueId, (also called the **accessId**), of the user who is logging into the front-end application. When this token is received by the downstream server it attempts to decrypt the token. If the LTPA keys are shared between the two servers, decryption succeeds and the accessId of the user is obtained from the token. The realm in the accessId is checked with the current realm that is used by the application. If the realms match, the LTPA token validation succeeds and it proceeds with the authorization. If the realms do not match, the token validation fails since the user from the foreign realm cannot be validated in the current realm of the application. If applications are not supposed to communicate with each other when using RMI/IIOP and the LTPA authentication mechanism, you do not to have to do anything further.

If you do want the cross realm communication to succeed when using RMI/IIOP and LTPA tokens, you must first establish trust between the realms involved, both for inbound and outbound communications.

For the server originating the request, its realm must have the realms that it can trust to send the token to. This is referred to as outboundTrustedRealms. For the server receiving the request, its realm needs to trust the realms that it can receive LTPA tokens from. This is referred to as inboundTrustedRealms.

Outbound trusted realms can be established using the addTrustedRealms command with the –communicationType option set to `outbound`. It can also be established in the administrative console by clicking **Trusted authentication realms - outbound** on the **CSIv2 outbound communications** panel.

Inbound trusted realms can be established using the same addTrustedRealms command task with the –communicationType option set to `inbound`. It can also be established by using the administrative console.

The figure later in this section shows the communication between applications that use different user realms (registries) using RMI/IIOP. In this example, application **app1** (for example, a **servlet**) is configured

to use the **realm1** user registry. The **app2** application (for example, an EJB) is configured to use the **realm2** user registry. The user (**user1**) initially logs into the servlet in **app1**, which then attempts to access an EJB in **app2**. The following must be set:

- In Domain1, **realm1** should trust **realm2** for the outbound communication.
- In Domain2, **realm2** should trust **realm1** for the inbound communication.
- The accessId for **user1** should be configured in the authorization table for **app2**.

When the LTPA token that contains the accessId of **user1** is received by **app2**, it decrypts the token. Both of the servers share the same LTPA keys. The LTPA token then ensures that the foreign realm is a trusted realm, and performs the authorization based on the accessId of **user1**. If security attribute propagation is not disabled, then the group information of **user1** is also propagated to **app2**. The groups can be used for the authorization check, provided that the authorization table contains the group information. You can associate a special subject, **AllAuthenticatedInTrustedRealms**, to the roles instead of adding individual users and groups to the authorization table.

If the applications in the previous example are deployed in different cells, you must do the following:

- Share the LTPA keys between the cells.
- Update the authorization table for **app2** with foreign users and groups accessIds by using the **wsadmin** utility. The administrative console does not have access to the realms outside of the scope of the cell.

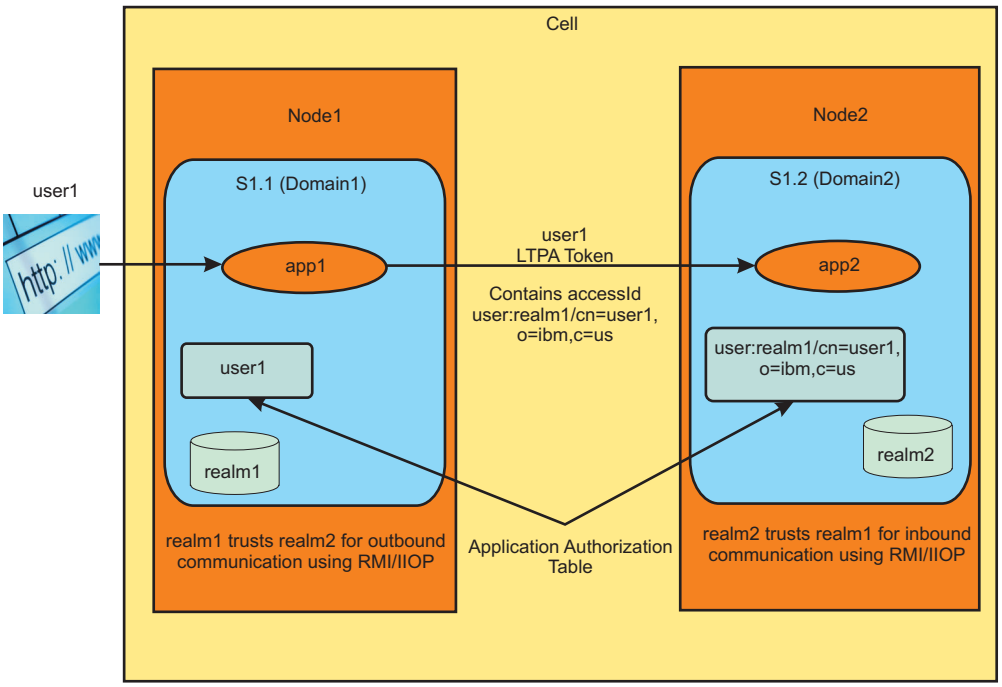


Figure 4. Cross realm communication in a multiple realm environment

Once trust has been established between the realms, when the server receives the LTPA token and the token is decrypted, it checks to see if the foreign realm is in its inbound trusted realms list. If it is trusted, the authentication succeeds. However, since it is a foreign realm, it does not go search the user registry to gather information about the user. Whatever information is in the LTPA token is used to authorize the user.

The only information in the LTPA token is the unique id of the user. This unique id of the user should exist in the authorization table for this application. If it does, authorization succeeds. However, if attribute propagation is enabled, additional authorization attributes (groups that this user belongs to) for the user

are sent from the originating server to the receiving server. These additional attributes are used to make the access decisions. If the groups information exists in the propagation tokens it is used when making the authorization decision.

As previously mentioned, the information about the users and or the groups from the trusted realms should exist in the authorization table of the receiving application. Specifically, the accessId of the users and or groups should exist in the binding file of the application. This must be the case when the application is deployed. In the administrative console, when an application is deployed in a domain you can add the accessIds of the users and groups from any of its trusted realms to the authorization table.

You also have an option to associate a special subject, **AllAuthenticatedInTrustedRealms**, to the roles instead of adding individual users and groups. This is similar to the **AllAuthenticated** special subject that is currently supported. The difference is that the **AllAuthenticated** special subject refers to users in the same realm as the application while the **AllAuthenticatedInTrustedRealms** special subject applies to all of the users in the trusted realms and in the realm of the application.

You can associate the accessId by using the $AdminApp install script. Because the accessId takes a unique format, use the command task **listRegistryUsers** with **displayAccessIds** set to `true`. If an invalid name or format is entered in this field, the authorization fails.

User and group information from the trusted realms is obtained by the deployment manager since it has access to all of the user registry configurations in all domains. However, in certain situations it is not possible to obtain the users and group information.

For example, if a server hosted on an external node is using localOS as the registry for its domain, the deployment manager cannot obtain the users and groups information unless it is running in the same operating system setup. The external operating system should be contacted to obtain this information. This can be done by directly invoking the registry in the server associated with that domain. The servers associated with the domain have to be started for this to work. You also must set the property, **com.ibm.websphere.allowRegistryLookupOnProcess**, to `true` in the top-level security custom properties. When this property is set, the deployment manager code searches one of the servers that is associated with the security domain and obtains the users and groups information directly from it. This is possible by calling an MBean in one of the servers.

If the MBean in any of the servers that are using that domain cannot be accessed, the administrative console displays a panel where you can enter the user and accessId information manually for each user and group. It is important that the correct accessId format be entered in this field. The accessId format for the user is `user:realmName/userUniqueId`. The realmName is the name of the realm where the user resides, and the userUniqueId is the uniqueId that represents the user, depending on the registry that is used.

For example, for LDAP, the uniqueUserId is the Distinguished Name (DN), for the Windows localOS registry and is the SID of the user. For Unix platforms, it is the UID. For custom registries, it depends on the implementation.

Similarly, for groups, the accessId format is group:realmName/groupUniqueId. As previously mentioned, use the listRegistryUsers and listRegistryGroups command with the –displayAccessIds option set to `true` so that you can obtain the correct format for the domain or realm that you are interested in.

Once users and groups from the trusted realms or the AllAuthenticatedInTrustedRealms special subject is added to the authorization table of the application, it is ready to accept requests from other applications that are using any of its trusted realms. The LTPA token validation on the receiving server first checks to make sure that the realm is trusted. The authorization engine then checks to see if the external user and/or the groups or the **AllAuthenticatedInTrustedRealms** special subject are given access to the roles needed to access the resource. If true, access is granted.

Cross realm communication is only applicable when using the WebSphere built-in authorization. If you are using other authorization engines including SAF for z/OS, any cross realm authorization can be achieved by implementing custom login modules that map external users to users in its own repository.

Federating a node with security domains

When a security domain is configured in the base version and is federated to a cell, the security domain configured at the base version is also configured for that server in the cell. The same domain security configuration can be used by the server before and after the federation. If a base server is to be federated to a cell, the resource assigned to the security domain should be the server scope instead of the cell scope.

If the base server is expected to be registered with an Administrative Agent process, use the cell scope as the resource if the intention is to have all of the servers in the base profile use this security domain.

If during federation the security domain at the base already exists at the cell level, the **addNode** command fails. You can use the –`excludesecuritydomains` option not to include the security domain during federation.

When the federated node is removed from a cell, the resources in that node should be removed from the security domains. If security domains have clusters associated with them that span nodes, the nodes are not removed. You can always remove resources from the security domains or any domains that are not used by using scripting commands or the administrative console.

Security domains in a mixed-version environment

You should create security domains once all of the nodes have been migrated to the latest version. This is especially true if there is a need to associate the cell with a domain. However, if you want to create security domains in a mixed- version environment, be aware of the following:

- If a cell-wide domain is created in a mixed version setup, a domain called **PassThroughToGlobalSecurity** is created automatically. All mixed clusters are assigned to this domain at the time of the creation of the cell-wide domain. This **PassThroughToGlobalSecurity** domain is special in the sense that attributes cannot be added to it, only resources can be assigned to it.

  All resources assigned to the **PassThroughToGlobalSecurity** domain use the global security configuration information. Whenever a node in the mixed version setup is migrated to the latest version, the servers and clusters in these nodes are added to this domain. Applications in all of the servers and clusters in these nodes do not use the cell-wide domain; they instead use the global security configuration before and after migration.

  If any of these servers need to use the cell-wide domain, you must remove these resources from this **PassThroughToGlobalSecurity** domain. New servers and clusters that are created in the migrated node use the cell-wide domain, not the **PassThroughToGlobalSecurity** domain. As a result, you have a mix of servers and clusters, some of them using global security configuration and some using the cell-wide domain.
- Once a cell-wide domain is created, adding any old version cluster members to a WebSphere Application Server Version 8.5 cluster is restricted since this action makes it a mixed cluster. This restriction also holds true when a WebSphere Application Server Version 8.5 cluster is associated with a domain. and a previous version cluster member is added to this cluster. This restriction is needed to avoid associating a security domain to a mixed cluster.
- If possible, you should create a cell-wide domain after all of the nodes have been migrated. In this case, the cell-wide domain is applicable to the entire cell and not just to parts of it. This also eliminates the need to create the **PassThroughToGlobalSecurity** domain and the mixed cluster scenario with security domains.

Modifying security domains

Use the administrative console tasks or scripting commands to modify security domains. For a complete list of administrative tasks and scripting commands, see the links in "Related tasks" at the bottom of this document.

Once a security domain is created and associated to a set of scopes, the servers associated with this new domain must be restarted. After the restart, the applications in the scopes associated with the new domain use the security attributes defined in the domain.

Changes to any of the domain attributes requires the restart of all of the scopes assigned to it. If new scopes are added they also need to be restarted. Any modifications to the domain configuration, either to the security attributes or to the scopes, has impacts on those applications that are using the domain configuration.

Before you make modifications to an existing domain, consider the following potential impacts. For example, if a user registry that is configured at a domain is removed, and the servers restarted, the user registry from the cell-wide domain (if one is defined), or the global security configuration is then used. This can impact application authentication and authorization. Users and groups associated with an application might no longer be valid in the new registry. Another example to consider is when JAAS configurations are removed from a domain. Applications that rely on this are no longer be able to use the JAAS configurations. Whenever a security configuration is changed it might impact your applications, so all security configuration changes should be made with the utmost care.

Creating new multiple security domains

You can create multiple security domains in your configuration. By creating multiple security domains, you can configure different security attributes for administrative and user applications within a cell environment.

Before you begin

Only users assigned to the administrator role can create new multiple security domains. Enable global security in your environment before creating new multiple security domains.

Read about “Multiple security domains” on page 126 for a better understanding of what multiple security domains are and how they are supported in this version of WebSphere Application Server.

About this task

Security domains provide a mechanism to use different security settings for administrative applications and user applications. They also provide the ability to support multiple security settings so different applications can use different security attributes like user registry or login configurations.

Use multiple security domains to achieve the following goals:

- Configure different security attributes for administrative and user applications within a cell
- Consolidate server configurations by managing different security configurations within a cell
- Restrict access between applications with different user registries, or configure trust relationships between applications to support communication across registries

Perform the following steps to create a new security domain using the administrative console:

Procedure

1. Click **Security > Security domains**.
2. On the Security domains collection page, click **New**.

3. Specify a unique name for the domain. A domain name must be unique within a cell and cannot contain an invalid character. This field is required.
4. Specify a unique description for the domain. After you click **Apply** you are returned to the Security domains detail page
5. Under Assigned Scopes, assign the security domain to the entire cell or select the specific servers, clusters, and service integration buses to include in the security domain.
6. Customize your security configuration by specifying security attributes for your new domain and by assigning it to cell resources.

   You can change security attributes such as the following:

   Application Security

   Specifies the settings for application security and Java 2 security. You can use the global security settings or customize the settings for a domain.

   Select **Enable application security** to enable or disable security this choice for user applications. When this selection is disabled, all of the EJBs and web applications in the security domain are no longer protected. Access is granted to these resources without user authentication. When you enable this selection, the J2EE security is enforced for all of the EJBs and web applications in the security domain. The J2EE security is only enforced when Global Security is enabled in the global security configuration, (that is, you cannot enable application security without first enabling Global Security at the global level).

   Java 2 Security

   Select **Java 2 security** to enable or disable Java 2 security at the domain level. This choice enables or disables Java 2 security at the process (JVM) level so that all applications (both administrative and user) can enable or disable Java 2 security.

   User realm

   This section enables you to configure the user registry for the security domain. You can separately configure any registry that is used at the domain level. Read about “Multiple security domains” on page 126 for more information.

   Trust association

   When you configure the trust association interceptor (TAI) at a domain level, the interceptors configured at the global level are copied to the domain level for convenience. You can modify the interceptor list at the domain level to fit your needs. Only configure those interceptors that are to be used at the domain level.

   SPNEGO Web Authentication

   The SPNEGO web authentication, which enables you to configure SPNEGO for web resource authentication, can be configured at the domain level.

   Note: In WebSphere Application Server Version 6.1, a TAI that uses the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) to securely negotiate and authenticate HTTP requests for secured resources was introduced. This function was deprecated in WebSphere Application Server Version 7.0. SPNEGO web authentication has taken its place to provide dynamic reload of the SPNEGO filters and to enable fallback to the application login method.

   RMI/IIOP Security

   The RMI/IIOP security attribute refers to the CSIv2 (Common Secure Interoperability version 2) protocol properties. When you configure these attributes at the domain level, the RMI/IIOP security configuration at the global level is copied for convenience.

   You can change the attributes that need to be different at the domain level. The Transport layer settings for CSIv2 inbound communications should be the same for both the global and the domain levels. If they are different, the domain level attributes are applied to all of the application in the process.

JAAS application logins

Specifies the configuration settings for the Java Authentication and Authorization Service (JAAS) application logins. You can use the global security settings or customize the settings for a domain.

The JAAS application logins, the JAAS system logins, and the JAAS J2C authentication data aliases can all be configured at the domain level. Be default, all of the applications in the system have access to the JAAS logins configured at the global level. The security runtime first checks for the JAAS logins at the domain level. If it does not find them, it then checks for them in the global security configuration. Configure any of these JAAS logins at a domain only when you need to specify a login that is used exclusively by the applications in the security domain.

JAAS system logins

Specifies the configuration settings for the JAAS system logins. You can use the global security settings or customize the configuration settings for a domain.

The JAAS application logins, the JAAS system logins, and the JAAS J2C authentication data aliases can all be configured at the domain level. Be default, all of the applications in the system have access to the JAAS logins configured at the global level. The security runtime first checks for the JAAS logins at the domain level. If it does not find them, it then checks for them in the global security configuration. Configure any of these JAAS logins at a domain only when you need to specify a login that is used exclusively by the applications in the security domain.

Note: For both JAAS application logins and JAAS system logins, the collections are not populated until one is created first. You can do this by selecting **customize for this domain** under JAAS application logins or JAAS system logins and then by selecting **Apply** or **OK**.

JAAS J2C authentication

Specifies the configuration settings for the JAAS J2C authentication data. You can use the global security settings or customize the settings for a domain.

The JAAS application logins, the JAAS system logins, and the JAAS J2C authentication data aliases can all be configured at the domain level. Be default, all of the applications in the system have access to the JAAS logins configured at the global level. The security runtime first checks for the JAAS logins at the domain level. If it does not find them, it then checks for them in the global security configuration. Configure any of these JAAS logins at a domain only when you need to specify a login that is used exclusively by the applications in the security domain.

Java Authentication SPI (JASPI)

Specifies the configuration settings for a Java Authentication SPI (JASPI) authentication provider. You can use the global security settings or customize the settings for a domain. To configure JASPI authentication providers for a domain, select **Customize for this domain** and then enable JASPI. Select **Providers** to define providers for the domain.

Note: The JASPI authentication provider can be enabled with providers configured at the domain level. By default, all of the applications in the system have access to the JASPI authentication providers configured at the global level. The security runtime first checks for the JASPI authentication providers at the domain level. If it does not find them, it then checks for them in the global security configuration. Configure JASPI authentication providers at a domain only when the provider is to be used exclusively by the applications in that security domain.

Authentication Mechanism Attributes

Specifies the various cache settings that need to applied at the domain level.

Select **Authentication cache settings** to specify your authentication cache settings. The configuration specified on this panel is applied only to this domain.

Select **LTPA Timeout** to configure a different LTPA timeout value at the domain level. The default timeout value is 120 minutes, which is set at the global level. If the LTPA timeout is set at the domain level, any token that is created in the security domain when accessing user applications is created with this expiration time.

When **Use realm-qualified user names** is enabled, user names returned by methods such as `getUserPrincipal( )` are qualified with the security realm (user registry) used by applications in the security domain.

Authorization Provider

You can configure an external third party JACC (Java Authorization Contract for Containers) provider at the domain level. Tivoli Access Manager's JACC provider can only be configured at the global level. Security domains can still use it if they do not override the authorization provider with another JACC provider or with the built-in native authorization.

Custom properties

Set custom properties at the domain level that are either new or different from those at the global level. By default, all of the custom properties at the global security configuration can be accessed by all of the applications in the cell. The security runtime code first checks for the custom property at the domain level. If it does not find it, it then attempts to obtain the custom property from the global security configuration.

7. Click **Apply**.
8. After you have saved your configuration changes, restart the server for your changes to take effect.

Deleting multiple security domains

You can delete multiple security domains from your configuration. You must remove the resources assigned to the security domains before deleting them. Only remove those security domains that are not needed in your security configuration.

Before you begin

Only users assigned to the administrator role can delete security domains. Enable global security in your environment before deleting security domains.

Read about “Multiple security domains” on page 126 for a better understanding of what multiple security domains are and how they are supported in this version of WebSphere Application Server.

About this task

Security domains provide a mechanism to use different security settings for administrative applications and user applications. They also provide the ability to support multiple security settings so different applications can use different security attributes like user registry or login configurations.

Perform the following steps to delete an existing security domain using the administrative console:

Note: Only delete the security domains after first removing any resources associated with them. The servers impacted should be restarted.

Procedure

1. Click **Security > Security domains**.
2. On the Security domains collection page, select a domain to delete.
3. Click **Delete**.

Copying multiple security domains

You can copy selected multiple security domains from the domain collection to create a new domain. This is useful if you want to create a domain that is similar to a previous domain. However, you might want to make a few slight adjustments. When copying an existing domain, you must supply a unique domain name for the new one.

Before you begin

Only users assigned to the administrator role can copy or create new multiple security domains. Enable global security in your environment before copying multiple security domains.

Read about "Multiple security domains" on page 126 for a better understanding of what multiple security domains are and how they are supported in this version of WebSphere Application Server.

About this task

Security domains provide a mechanism to use different security settings for administrative applications and user applications. They also provide the ability to support multiple security settings so different applications can use different security attributes like user registry or login configurations.

Use multiple security domains to achieve the following goals:

- Configure different security attributes for administrative and user applications within a cell
- Consolidate server configurations by managing different security configurations within a cell
- Restrict access between applications with different user registries, or configure trust relationships between applications to support communication across registries

Perform the following steps to copy an existing security domain using the administrative console:

Procedure

1. Click **Security > Security domains**.
2. Optional: From Preferences, you can select the maximum number of rows to display when the domain collection is large. The default number of rows is 20. Rows that exceed that number appear on subsequent pages.
3. Select a domain to copy.
4. Click **Copy Selected Domain...** to copy an existing domain from the collection. You can optionally select **Copy Global Security..** to copy an existing domain and have it maintain its global security settings (collection selections are ignored). A new domain name is also required if you choose this option.
5. Specify a unique name for the domain. This field is required. A domain name must be unique within a cell and cannot contain an invalid character.
6. Specify a unique description for the domain.
7. Click **Apply**. After you click **Apply** you are returned to the Security domains detail page
8. Under Assigned Scopes, assign the security domain to the entire cell or select the specific servers, clusters, and service integration buses to include in the security domain.
9. Customize your security configuration by specifying security attributes for your new domain and by assigning it to cell resources.

   You can change security attributes such as the following:

   Application Security

   Specifies the settings for application security and Java 2 security. You can use the global security settings or customize the settings for a domain.

Select **Enable application security** to enable or disable security this choice for user applications. When this selection is disabled, all of the EJBs and web applications in the security domain are no longer protected. Access is granted to these resources without user authentication. When you enable this selection, the J2EE security is enforced for all of the EJBs and web applications in the security domain. The J2EE security is only enforced when Global Security is enabled in the global security configuration, (that is, you cannot enable application security without first enabling Global Security at the global level).

Java 2 Security

Select **Java 2 security** to enable or disable Java 2 security at the domain level. This choice enables or disables Java 2 security at the process (JVM) level so that all applications (both administrative and user) can enable or disable Java 2 security.

User realm

This section enables you to configure the user registry for the security domain. You can separately configure any registry that is used at the domain level. Read about "Multiple security domains" on page 126 for more information.

Trust association

When you configure the trust association interceptor (TAI) at a domain level, the interceptors configured at the global level are copied to the domain level for convenience. You can modify the interceptor list at the domain level to fit your needs. Only configure those interceptors that are to be used at the domain level.

SPNEGO Web Authentication

The SPNEGO web authentication, which enables you to configure SPNEGO for web resource authentication, can be configured at the domain level.

Note: In WebSphere Application Server Version 6.1, a TAI that uses the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) to securely negotiate and authenticate HTTP requests for secured resources was introduced. This function was deprecated in WebSphere Application Server 7.0. SPNEGO web authentication has taken its place to provide dynamic reload of the SPNEGO filters and to enable fallback to the application login method.

RMI/IIOP Security

The RMI/IIOP security attribute refers to the CSIv2 (Common Secure Interoperability version 2) protocol properties. When you configure these attributes at the domain level, the RMI/IIOP security configuration at the global level is copied for convenience.

You can change the attributes that need to be different at the domain level. The Transport layer settings for CSIv2 inbound communications should be the same for both the global and the domain levels. If they are different, the domain level attributes are applied to all of the application in the process.

JAAS application logins

Specifies the configuration settings for the Java Authentication and Authorization Service (JAAS) application logins. You can use the global security settings or customize the settings for a domain.

The JAAS application logins, the JAAS system logins, and the JAAS J2C authentication data aliases can all be configured at the domain level. Be default, all of the applications in the system have access to the JAAS logins configured at the global level. The security runtime first checks for the JAAS logins at the domain level. If it does not find them, it then checks for them in the global security configuration. Configure any of these JAAS logins at a domain only when you need to specify a login that is used exclusively by the applications in the security domain.

JAAS system logins

Specifies the configuration settings for the JAAS system logins. You can use the global security settings or customize the configuration settings for a domain.

The JAAS application logins, the JAAS system logins, and the JAAS J2C authentication data aliases can all be configured at the domain level. Be default, all of the applications in the system have access to the JAAS logins configured at the global level. The security runtime first checks for the JAAS logins at the domain level. If it does not find them, it then checks for them in the global security configuration. Configure any of these JAAS logins at a domain only when you need to specify a login that is used exclusively by the applications in the security domain.

JAAS J2C authentication

Specifies the configuration settings for the JAAS J2C authentication data. You can use the global security settings or customize the settings for a domain.

The JAAS application logins, the JAAS system logins, and the JAAS J2C authentication data aliases can all be configured at the domain level. Be default, all of the applications in the system have access to the JAAS logins configured at the global level. The security runtime first checks for the JAAS logins at the domain level. If it does not find them, it then checks for them in the global security configuration. Configure any of these JAAS logins at a domain only when you need to specify a login that is used exclusively by the applications in the security domain.

Java Authentication SPI (JASPI)

Specifies the configuration settings for a Java Authentication SPI (JASPI) authentication provider. You can use the global security settings or customize the settings for a domain. To configure JASPI authentication providers for a domain, select **Customize for this domain** and then enable JASPI. Select **Providers** to define providers for the domain.

Note: The JASPI authentication provider can be enabled with providers configured at the domain level. By default, all of the applications in the system have access to the JASPI authentication providers configured at the global level. The security runtime first checks for the JASPI authentication providers at the domain level. If it does not find them, it then checks for them in the global security configuration. Configure JASPI authentication providers at a domain only when the provider is to be used exclusively by the applications in that security domain.

Authentication Mechanism Attributes

Specifies the various cache settings that need to applied at the domain level.

Select **Authentication cache settings** to specify your authentication cache settings. The configuration specified on this panel is applied only to this domain.

Select **LTPA Timeout** to configure a different LTPA timeout value at the domain level. The default timeout value is 120 minutes, which is set at the global level. If the LTPA timeout is set at the domain level, any token that is created in the security domain when accessing user applications is created with this expiration time.

When **Use realm-qualified user names** is enabled, user names returned by methods such as `getUserPrincipal( )` are qualified with the security realm (user registry) used by applications in the security domain.

Authorization Provider

You can configure an external third party JACC (Java Authorization Contract for Containers) provider at the domain level. Tivoli Access Manager's JACC provider can only be configured at the global level. Security domains can still use it if they do not override the authorization provider with another JACC provider or with the built-in native authorization.

Custom properties

Set custom properties at the domain level that are either new or different from those at the global level. By default, all of the custom properties at the global security configuration can be accessed by all of the applications in the cell. The security runtime code first checks for the custom property at the domain level. If it does not find it, it then attempts to obtain the custom property from the global security configuration.

10. Click **Apply**.
11. After you have saved your configuration changes, restart the server for your changes to take effect.

Configuring inbound trusted realms for multiple security domains

You can configure which realms to grant inbound trust to for multiple security domains. The trust relationship between realms is used when communicating with Lightweight Third-Party Authentication (LTPA) tokens. Once a LTPA token is decrypted by the receiving server, the realm in the token is checked to see if it is trusted. If it is not, the validation of the token fails. A *realm* represents a user registry in WebSphere Application Server.

Before you begin

For information on cross realm communications, read the section in “Multiple security domains” on page 126.

Only users assigned to the administrator role can configure multiple security domains. Enable global security in your environment before configuring multiple security domains.

Perform the following steps to grant inbound trusted realms for multiple security domains using the administrative console:

Procedure

1. Click **Security > Security domains**.
2. Select a domain to edit or create a new one. Under Security Attributes, click **User realm**.
3. Click **Customize for this domain**.
4. Under Related Items, select **Trusted authentication realms - inbound**.
5. Select **Trust all realms (including those external to this cell)** or **Trust realms as indicated below**. If Kerberos authentication is enabled, and you have cross realms or trusted realms, you must add the Kerberos trusted realm by selecting **Trust realms as indicated below**.
6. Click **Apply.**

What to do next

The realms you selected to trust accept messages from other trusted realms but do not accept messages from untrusted realms. Select **Add External Realm** to add trust for realms that are external to this cell.

Configure security domains

Use this page to configure the security attributes of a domain and to assign the domain to cell resources. For each security attribute, you can use the global security settings or customize settings for the domain.

To view this administrative console page, click **Security > Security domains**. On the Security domains collection page, select an existing domain to configure, create a new one, or copy an existing domain.

Read about “Multiple security domains” on page 126 for a better understanding of what multiple security domains are and how they are supported in this version of WebSphere Application Server.

Name
Specifies a unique name for the domain. This name can not be edited after the initial submission.

A domain name must be unique within a cell and cannot contain an invalid character.

Description
Specifies a description for the domain.

Assigned Scopes
Select to display the cell topology. You can assign the security domain to the entire cell or select the specific clusters, nodes and service integration buses to include in the security domain.

If you select **All scopes**, the entire cell topology is displayed.

If you select **Assigned scopes**, the cell topology is displayed with those servers and clusters that are assigned to the current domain.

The name of an explicitly assigned domain appears next to any resource. Checked boxes indicate resources that are currently assigned to the domain. You also can select other resources and click **Apply** or **OK** to assign them to the current domain.

A resource that is not checked (disabled) indicates that it is not assigned to the current domain and must be removed from another domain before it can be enabled for the current domain.

If a resource does not have an explicitly-assigned domain, it uses the domain assigned to the cell. If no domain is assigned to the cell, then the resource uses global settings.

Cluster members cannot be individually assigned to domains; the enter cluster uses the same domain.

Application Security:
Select **Enable application security** to enable or disable security for user applications. You can use the global security settings or customize the settings for a domain.

When this selection is disabled, all of the EJBs and web applications in the security domain are no longer protected. Access is granted to these resources without user authentication. When you enable this selection, the J2EE security is enforced for all of the EJBs and web applications in the security domain. The J2EE security is only enforced when Global Security is enabled in the global security configuration, (that is, you cannot enable application security without first enabling Global Security at the global level).

Enable application security
Enables security for the applications in your environment. This type of security provides application isolation and requirements for authenticating application users

In previous releases of WebSphere Application Server, when a user enabled global security, both administrative and application security were enabled. In WebSphere Application Server Version 6.1, the previous notion of global security were split into administrative security and application security, each of which you can enable separately.

As a result of this split, WebSphere Application Server clients must know whether application security is disabled at the target server. Administrative security is enabled, by default. Application security is disabled, by default. To enable application security, you must enable administrative security. Application security is in effect only when administrative security is enabled.

When this selection is disabled, all of the EJBs and web applications in the security domain are no longer protected. Access is granted to these resources without user authentication. When you enable this selection, the J2EE security is enforced for all of the EJBs and web applications in the security domain. The J2EE security is only enforced when Global Security is enabled in the global security configuration, (that is, you cannot enable application security without first enabling Global Security at the global level).

Java 2 security:

Select **Use Java 2 security** to enable or disable Java 2 security at the domain level or to assign or add properties related to Java 2 security. You can use the global security settings or customize the settings for a domain.

This choice enables or disables Java 2 security at the process (JVM) level so that all applications (both administrative and user) can enable or disable Java 2 security.

Use global security settings

Select to specify the global security settings that are being used.

Customize for this domain

Select to specify the settings that are defined in the domain, such as options to enable application and Java 2 security and to use realm-qualified authentication data.

Use Java 2 security to restrict application access to local resources

Select to specify whether to enable or disable Java 2 security permission checking. By default, access to local resources is not restricted. You can choose to disable Java 2 security, even when application security is enabled.

When the **Use Java 2 security to restrict application access to local resources** option is enabled and if an application requires more Java 2 security permissions than are granted in the default policy, the application might fail to run properly until the required permissions are granted in either the `app.policy` file or the `was.policy` file of the application. AccessControl exceptions are generated by applications that do not have all the required permissions.

Warn if applications are granted custom permissions

Specifies that during application deployment and application start, the security runtime issues a warning if applications are granted any custom permissions. Custom permissions are permissions that are defined by the user applications, not Java API permissions. Java API permissions are permissions in the `java.*` and `javax.*` packages.

The application server provides support for policy file management. A number of policy files are available in this product, some of them are static and some of them are dynamic. Dynamic policy is a template of permissions for a particular type of resource. No code base is defined and no relative code base is used in the dynamic policy template. The real code base is dynamically created from the configuration and run-time data. The `filter.policy` file contains a list of permissions that you do not want an application to have according to the J2EE 1.4 specification.

Important: You cannot enable this option without enabling the **Use Java 2 security to restrict application access to local resources** option.

Restrict access to resource authentication data

This option is disabled if Java 2 security has not been enabled.

Consider enabling this option when both of the following conditions are true:

- Java 2 security is enforced.

- The application code is granted the accessRuntimeClasses WebSphereRuntimePermission permission in the `was.policy` file found within the application enterprise archive (EAR) file. For example, the application code is granted the permission when the following line is found in your `was.policy` file:

```
permission com.ibm.websphere.security.WebSphereRuntimePermission "accessRuntimeClasses";
```

The **Restrict access to resource authentication data** option adds fine-grained Java 2 security permission checking to the default principal mapping of the WSPrincipalMappingLoginModule implementation. You must grant explicit permission to Java 2 Platform, Enterprise Edition (J2EE) applications that use the WSPrincipalMappingLoginModule implementation directly in the Java Authentication and Authorization Service (JAAS) login when **Use Java 2 security to restrict application access to local resources** and the **Restrict access to resource authentication data** options are enabled.

Information	Value
Default:	Disabled

User Realm:

This section enables you to configure the user registry for the security domain. You can separately configure any registry that is used at the domain level.

When configuring a registry at the domain level you can choose to define your own realm name for the registry. The realm name distinguishes one user registry from another. The realm name is used in multiple places – in the Java client login panel to prompt the user, in the authentication cache, and when using native authorization.

At the global configuration level, the system creates the realm for the user registry. In previous releases of WebSphere Application Server, only one user registry is configured in the system. When you have multiple security domains you can configure multiple registries in the system. For the realms to be unique in these domains, configure your own realm name for a security domain. You also can choose the system to create a unique realm name if it is certain to be unique. In the latter case, the realm name is based on the registry that is being used.

Trust Association:

Select to specify the settings for the trust association. Trust association is used to connect reversed proxy servers to the application servers.

Trust association enables the integration of IBM WebSphere Application Server security and third-party security servers. More specifically, a reverse proxy server can act as a front-end authentication server while the product applies its own authorization policy onto the resulting credentials that are passed by the proxy server.

Tivoli Access Manager's trust association interceptors can only be configured at the global level. The domain configuration can also use them, but cannot have a different version of the trust association interceptor. Only one instance of Tivoli Access Manager's trust association interceptors can exist in the system.

Note: The use of trust association interceptors (TAIs) for Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) authentication is deprecated. The SPNEGO web authentication panels provide a much easier way to configure SPNEGO.

Interceptors

Select to access or to specify the trust information for reverse proxy servers.

Enable trust association

Select to enable the integration of IBM WebSphere Application Server security and third-party security servers. More specifically, a reverse proxy server can act as a front-end authentication server while the product applies its own authorization policy onto the resulting credentials that are passed by the proxy server.

SPNEGO Web Authentication:

Specifies the settings for Simple and Protected GSS-API Negotiation (SPNEGO) as the web authentication mechanism.

The SPNEGO web authentication, which enables you to configure SPNEGO for web resource authentication, can be configured at the domain level.

Note: In WebSphere Application Server Version 6.1, a TAI that uses the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) to securely negotiate and authenticate HTTP requests for secured resources was introduced. In WebSphere Application Server 7.0, this function is deprecated. SPNEGO web authentication has taken its place to provide dynamic reload of the SPNEGO filters and to enable fallback to the application login method.

RMI/IIOP Security:

Specifies the settings for Remote Method Invocation over the Internet Inter-ORB Protocol (RMI/IIOP).

An Object Request Broker (ORB) manages the interaction between clients and servers, using the Internet InterORB Protocol (IIOP). It enables clients to make requests and receive responses from servers in a network-distributed environment.

When you configure these attributes at the domain level, the RMI/IIOP security configuration at the global level is copied for convenience. You can change the attributes that need to be different at the domain level. The Transport layer settings for CSIv2 inbound communications should be the same for both the global and the domain levels. If they are different, the domain level attributes are applied to all of the applications in the process.

When a process communicates with another process with a different realm, the LTPA authentication and the propagation tokens are propagated to the downstream server unless that server is listed in the outbound trusted realms list. This can be done using the **Trusted authentication realms – outbound** link on the **CSIv2 outbound communication** panel.

CSIv2 inbound communications

Select to specify authentication settings for requests that are received and transport settings for connections that are accepted by this server using the Object Management Group (OMG) Common Secure Interoperability (CSI) authentication protocol.

WebSphere Application Server enables you to specify Internet Inter-ORB Protocol (IIOP) authentication for both inbound and outbound authentication requests. For inbound requests, you can specify the type of accepted authentication, such as basic authentication.

CSIv2 outbound communications

Select to specify authentication settings for requests that are sent and transport settings for connections that are initiated by the server using the Object Management Group (OMG) Common Secure Interoperability (CSI) authentication protocol.

WebSphere Application Server enables you to specify Internet Inter-ORB Protocol (IIOP) authentication for both inbound and outbound authentication requests. For outbound requests, you can specify properties such as type of authentication, identity assertion or login configurations that are used for requests to downstream servers.

JAAS Application logins

Select to define login configurations that are used by JAAS.

The JAAS application logins, the JAAS system logins, and the JAAS J2C authentication data aliases can all be configured at the domain level. By default, all of the applications in the system have access to the JAAS logins configured at the global level. The security runtime first checks for the JAAS logins at the domain level. If it does not find them, it then checks for them in the global security configuration. Configure any of these JAAS logins at a domain only when you need to specify a login that is used exclusively by the applications in the security domain.

For JAAS and custom properties only, once global attributes are customized for a domain they can still be used by user applications.

Do not remove the ClientContainer, DefaultPrincipalMapping, and WSLogin login configurations because other applications might use them. If these configurations are removed, other applications might fail.

Use global and domain-specific logins

Select to specify the settings that are defined in the domain, such as options to enable application and Java 2 security and to use realm-qualified authentication data.

JAAS System Logins:

Specifies the configuration settings for the JAAS system logins. You can use the global security settings or customize the configuration settings for a domain.

System Logins

Select to define the JAAS login configurations that are used by system resources, including the authentication mechanism, principal mapping, and credential mapping

JAAS J2C Authentication Data:

Specifies the settings for the JAAS J2C authentication data. You can use the global security settings or customize the settings for a domain.

Java 2 Platform, Enterprise Edition (J2EE) Connector authentication data entries are used by resource adapters and Java DataBase Connectivity (JDBC) data sources.

Use global and domain-specific entries

Select to specify the settings that are defined in the domain, such as options to enable application and Java 2 security and to use realm-qualified authentication data.

Java Authentication SPI (JASPI)

Specifies the configuration settings for a Java Authentication SPI (JASPI) authentication provider and associated authentication modules. You can use the global security settings or customize the settings for a domain. To configure JASPI authentication providers for a domain, select **Customize for this domain** and then you can enable JASPI. Select **Providers** to create or to edit a JASPI authentication provider.

Note: The JASPI authentication provider can be enabled with providers configured at the domain level. By default, all of the applications in the system have access to the JASPI authentication providers

configured at the global level. The security runtime first checks for the JASPI authentication providers at the domain level. If it does not find them, it then checks for them in the global security configuration. Configure JASPI authentication providers at a domain only when the provider is to be used exclusively by the applications in that security domain.

Authentication Mechanism Attributes:

Specifies the various cache settings that must be applied at the domain level.

- Authentication cache settings - use to specify your authentication cache settings. The configuration specified on this panel is applied only to this domain.
- LTPA Timeout - You can configure a different LTPA timeout value at the domain level. The default timeout value is 120 minutes, which is set at the global level. If the LTPA timeout is set at the domain level, any token that is created in the security domain when accessing user applications is created with this expiration time.
- Use realm-qualified user names - When this selection is enabled, user names returned by methods such as `getUserPrincipal( )` are qualified with the security realm (user registry) used by applications in the security domain.

Authorization Provider:

Specifies the settings for the authorization provider. You can use the global security settings or customize the settings for a domain.

You can configure an external third party JACC (Java Authorization Contract for Containers) provider at the domain level. Tivoli Access Manager's JACC provider can only be configured at the global level. Security domains can still use it if they do not override the authorization provider with another JACC provider or with the built-in native authorization.

Select either the **Default authorization** or **External authorization using a JAAC provider**. The **Configure** button is only enabled when **External authorization using a JAAC provider** is selected.

Custom properties

Select to specify name-value pairs of data, where the name is a property key and the value is a string.

Set custom properties at the domain level that are either new or different from those at the global level. By default, all of the custom properties at the global security configuration can be accessed by all of the applications in the system. The security runtime code first checks for the custom property at the domain level. If it does not find it, it then attempts to obtain the custom property from the global security configuration.

Web Services Bindings

Click **Default policy set bindings** to set the domain default provider and client bindings.

External realm name

Use this page to add a WebSphere Application Server realm that is external to this cell. The realm is initially not trusted. Use the Trusted authentication realms - inbound page to establish trust.

To view this administrative console page, click **Security > Security domains**. Select a domain to edit or create a new one. Under Security Attributes, click **User realm**. Click **Customize for this domain** and then select a **Realm type**. Click **Configure**. Under Related items, click **Trusted authentication realms - inbound** or **Trusted authentication realms - outbound**. Click **Add External Realm...**.

External realm name

Use to specify the name of the realm that is external to the list of realms that are available to receive trust.

Trust all realms

Use this page to configure which realms to grant inbound or outbound trust to.

The inbound trust is required to validate LTPA tokens that contain a foreign realm. The outbound trust is required to send the credential tokens to the trusted realms. For example, if an application using realmA needs to communicate using LTPA with an application using realmB, realmA should have realmB in its outbound trust list and realmB should have realmA in its inbound trust list.

To view this administrative console page, click **Security** > **Security domains**. Select a domain to edit or create a new one. Under Security Attributes, click **User realm**. Click **Customize for this domain**. Select a realm type and then click **Configure**.

Under Related items, click **Trusted authentication realms - inbound** or **Trusted authentication realms - outbound**.

Trust all realms (including those external to this cell)

Select to trust all of the realms listed on this page, including those external to the cell.

Trust realms as selected

Select to trust only those realms that you have selected from the list of realms that are available to receive inbound trust.

Add External Realm...

Select to add realms that are external to this cell to the list of realms that are available to receive inbound trust. When an external realm is added, it is trusted by default. If it is not trusted it is removed from the list.

Security domains collection

Security domains provide a mechanism to use different security settings for administrative applications and user applications. They also provide the ability to support multiple security settings so different application servers can use different security attributes like user registry or login configurations.

To view this administrative console page, click **Security > Security domains**.

Read about “Multiple security domains” on page 126 for a better understanding of what multiple security domains are and how they are supported in this version of WebSphere Application Server.

Maximum rows

Specifies the maximum number of rows that display when the collection is large. The rows that are not displayed appear on the next page.

The default is 20. Rows that exceed the maximum number display on subsequent pages.

Retain filter criteria

Specifies whether to use the same filter criteria entered in the show filter function to display this page the next time you visit it.

Copy selected domain

Select to copy a selected domain from the collection (a new name is required)

Copy global security

Select to create a domain with a copy of the global security settings (collection selections are ignored). A domain name is required.

Authentication cache settings

Use this page to specify your authentication cache settings.

To view this administrative console page, click **Security > Global security > Authentication cache settings**.

Enable authentication cache

Specifies whether to disable the authentication cache.

Leave the authentication cache enabled for performance reasons. However, you can disable the authentication cache for debug or measurement purposes. When this choice is disabled, the performance is impacted since whenever a user is authenticated the user registry is accessed to gather information about the user. New tokens are then created for the user.

Information	Value
Default:	Enabled

Cache timeout:

Specifies the time period at which the authenticated credential in the cache expires. Verify that this time period is less than the value for the **Timeout value for forwarded credentials between servers** field (the LTPA timeout).

If the application server infrastructure security is enabled, the security cache timeout can influence performance. The timeout setting specifies how often to refresh the security-related caches. Security information pertaining to beans, permissions, and credentials is cached. When the cache timeout expires, all cached information not accessed within the timeout period is purged from the cache. Subsequent requests for the information result in a database lookup. On occasion, acquiring the information requires invoking a Lightweight Directory Access Protocol (LDAP)-bind or native authentication. Both invocations are relatively costly operations for performance. Determine the best trade-off for the application by looking at usage patterns and security needs for the site.

You must consider the following effects of this value on your configuration:

- Larger authentication cache timeout values can increase the security risk. For example, you might revoke a user in the user registry or repository. However, the revoked user can log into the administrative console using the credential that is cached in the authentication cache until the cache is refreshed.
- Smaller authentication cache timeout values can affect performance. When this value is smaller, the application server accesses the user registry or repository more frequently.
- Larger numbers of entries in the authentication cache, which is due to an increased number of users, increases the memory usage by the authentication cache. Thus, the application server might slow down and affect performance.

You can limit the size of the authentication cache by setting the maximum cache size value. Set both the maximum cache size and the authentication cache timeout values to balance your security risk and performance needs.

The LTPA timeout value should not be set lower than the security cache timeout value. The LTPA timeout value should be set later than the ORB request timeout value. However, there is no relation between the security cache timeout value and the ORB request timeout value. For more information on the LTPA

timeout value, see the documentation about authentication mechanisms and expiration. For more information on the ORB request timeout value, see the documentation about the Object Request Broker service settings.

Information	Value
Default:	10 minutes

Initial cache size:

Specifies the initial size of the hash table caches.

A greater number of available hash values might decrease the occurrence of hash collisions. A hash collision results in a linear search for the hash bucket, which might decrease the retrieval time. If several entries compose a hash table cache, create a table with a larger capacity that supports more efficient hash entries instead of allowing automatic rehashing determine the growth of the table. Rehashing causes every entry to move each time.

Information	Value
Default:	50

Maximum cache size

Indicates the maximum size of the cache.

After this limit is reached, the least used entries are removed from the cache to make space for the new entries.

Information	Value
Default:	25000

Use basic authentication cache keys (password one-way hashed):

Caches the userName and the one-way hashed password as the key lookup in the cache.

Disable this only if you do not want this information to be stored in the cache. If this is disabled, every time a user logs in with userName and password, the user registry is accessed, which impacts performance.

Information	Value
Default:	True

Chapter 7. Authenticating users

The process of authenticating users involves a user registry and an authentication mechanism. Optionally, you can define trust between WebSphere Application Server and a proxy server, configure single sign-on capability, and specify how to propagate security attributes between application servers.

About this task

The following security topics are covered in this section:

User registries
For information on local operating system, Lightweight Directory Access Protocol (LDAP), custom user registries, and user repositories such as virtual member manager, see "Selecting a registry or repository."

Trust associations
For more information on trust associations, see "Trust associations" on page 364.

Single sign-on
For more information on single sign-on, see "Single sign-on for authentication using LTPA cookies" on page 370.

Security attribute propagation
For more information on propagation tokens, authorization tokens, single sign-on tokens, and authentication tokens, see "Security attribute propagation" on page 468.

The following information is covered in this section:

Procedure

- Configure a user registry. For more information, see "Selecting a registry or repository."
- Configure WebSEAL or a custom trust association interceptor. For more information see, "Integrating third-party HTTP reverse proxy servers" on page 364.
- Configure single sign-on. For more information, see "Implementing single sign-on to minimize web user authentications" on page 373.
- Propagate security attributes. For more information, see "Propagating security attributes among application servers" on page 473.
- Configure the authentication cache. For more information, see "Configuring the authentication cache" on page 485.

What to do next

After completing the configuring the authentication process, you must authorize access to resources. For more information, see Chapter 8, "Authorizing access to resources," on page 565.

Selecting a registry or repository

Information about users and groups reside in a user registry. In WebSphere Application Server, a user registry authenticates a user and retrieves information about users and groups to perform security-related functions, including authentication and authorization.

Before you begin

Note: During profile creation, either during installation or post-installation, administrative security is enabled by default. The file-based federated user repository is configured as the active user registry. Decide if you want a different user registry.

Before configuring the user registry or repository, decide which user registry or repository to use. You can configure one Active default registry for the Cell.

About this task

WebSphere Application Server provides implementations that support multiple types of registries and repositories including the local operating system registry, a stand-alone Lightweight Directory Access Protocol (LDAP) registry, a stand-alone custom registry, and federated repositories.

With WebSphere Application Server, a user registry or a repository, such as a federated repository, authenticates a user and retrieves information about users and groups to perform security-related functions including authentication and authorization.

With WebSphere Application Server, a user registry or repository is used for:

- Authenticating a user using basic authentication, identity assertion, or client certificates
- Retrieving information about users and groups to perform security-related administrative functions, such as mapping users and groups to security roles

In addition to local operating system, LDAP, and Federated repository registries, WebSphere Application Server also provides a plug-in to support any registry by using the custom registry feature. The custom registry feature enables you to configure any user registry that is not made available through the security configuration panels of the WebSphere Application Server.

Configuring the correct registry or repository is a prerequisite to assigning users and groups to roles for applications. When a user registry or repository is not configured, the local operating system registry is used by default. If your choice of user registry is not the local operating system registry, you need to first configure the registry or repository, which is normally done as part of enabling security, restart the servers, and then assign users and groups to roles for all your applications.

WebSphere Application Server supports the following types of user registries:

- Federated repository
- Local operating system

  Restriction: Configuring a transparent LDAP server under the local operating system registry and having authentication of users take place through that local operating system using LDAP is unsupported.
- Standalone Lightweight Directory Access Protocol (LDAP) registry
- Stand-alone custom registry

The UserRegistry interface is used to implement both the custom registry and the federated repository options for the user account repository. The interface is very helpful in situations where the current user and group information exists in some other formats, for example, a database, and cannot move to local operating system or LDAP registries. In such a case, you can implement the UserRegistry interface so that WebSphere Application Server can use the existing registry for all the security-related operations. The process of implementing a custom registry is a software implementation effort, and it is expected that the implementation does not depend on WebSphere Application Server resource management for its operation. For example, you cannot use an Application Server data source configuration; generally you must invoke database connections and dictate their behavior directly in your code.

Note: WebSphere Application Server has implemented a user registry proxy by using the UserRegistry interface. However, the return values are little different from the interface. For example, getUniqueUserId returns the uniqueID with the realm name wrapped. You cannot use the return value to pass to getUserSecurityName, as shown in the following example:

```
// Retrieves the default InitialContext for this server.
javax.naming.InitialContext ctx = new javax.naming.InitialContext();

// Retrieves the local UserRegistry object.
com.ibm.websphere.security.UserRegistry reg =
         (com.ibm.websphere.security.UserRegistry) ctx.lookup("UserRegistry");

// Retrieves the registry uniqueID based on the userName that is specified
     // in the NameCallback.
String uniqueid = reg.getUniqueUserId(userName);
// Strip the realm name and get real uniqueID
String uid = com.ibm.wsspi.security.token.WSSecurityPropagationHelper.getUserFromUniqueID (uniqueID);

// Retrieves the security name from the user registry based on the uniqueID.
String securityName = reg.getUserSecurityName(uid);
```

You can use a Service Provider Interface (SPI) for this parsing function.

After the applications are assigned users and groups and you need to change the user registries, delete all the users and groups, including any RunAs role, from the applications, and reassign them after changing the registry through the administrative console or by using wsadmin scripting. The following **wsadmin** command, which uses Jacl, removes all of the users and groups from any application:

```
$AdminApp deleteUserAndGroupEntries yourAppName
```

where *yourAppName* is the name of the application. Backing up the old application is advised before performing this operation. However, if both of the following conditions are true, you might be able to switch the registries without having to delete the users and groups information:

- All of the user and group names, including the password for the RunAs role users, in all of the applications match in both user registries.
- The application bindings file does not contain the access IDs which are unique for each user registry even for the same user or group name.

By default, an application does not contain access IDs in the bindings file. These IDs are generated when the applications start. However, if you migrated an existing application from an earlier release, or if you used the wsadmin script to add access IDs for the applications to improve performance, you have to remove the existing user and group information and add the information after configuring the new user registry.

For more information on updating access IDs, see updateAccess IDs in the Commands for the AdminApp object article.

Attention: WebSphere Application Server supports a variety of user registries and repositories on different operating systems. During the user authentication process, you might use non-alphanumeric characters in your user name or password. Restrictions on the use of these non-alphanumeric characters depends on both the underlying operating system and the user registry type. For more information on which non-alphanumeric characters are not supported, see your operating system and user registry or repository documentation.

AIX For example, the following characters are not supported in a user name value:

- `
- #
- =
- \
- :
- "
- ,
- /
- ?

- '
- A space character

For a comprehensive list of the non-alphanumeric characters that are not supported, see the IBM AIX operating system documentation.

HP-UX For example, the following characters are not supported in a user name value:
- `
- :
- "
- /
- A space character

Complete one of the following steps to configure your user registry:

Procedure
- "Configuring local operating system registries"
- "Configuring Lightweight Directory Access Protocol user registries" on page 170
- "Configuring stand-alone custom registries" on page 196.
- "Managing the realm in a federated repository configuration" on page 226

What to do next
1. If you are enabling security, make sure that you complete the remaining steps. Verify that the User account repository on the Global security panel is set to the appropriate registry or repository. As the final step, validate the user ID and the password by clicking **Apply** on the Global security panel. Save, stop and start all WebSphere Application Servers.
2. For any changes in user registry panels to be effective, you must validate the changes by clicking **Apply** on the Global security panel. After validation, save the configuration and stop and start all WebSphere Application Servers, including the cells, nodes and all of the application servers. To avoid inconsistencies between the WebSphere Application Server processes, make sure that any changes to the registry or repository are done when all of the processes are running. If any of the processes are down, force synchronization to make sure that the process can start later.

   If the server or servers start without any problems, the setup is correct.

Configuring local operating system registries
Use these steps to configure local operating system registries.

Before you begin

For detailed information about using the local operating system user registry, see "Local operating system registries" on page 164. These steps set up security based on the local operating system user registry on which WebSphere Application Server is installed.

For security purposes, the WebSphere Application Server provides and supports the implementation for Windows operating system registries, AIX, Solaris and multiple versions of Linux operating systems. The respective operating system application programming interface (API) are called by the product processes (servers) for authenticating a user and other security-related tasks (for example, getting user or group information). Access to these APIs are restricted to users who have special privileges. These privileges depend on the operating system and are described below.

In WebSphere Application Server Version 6.1, you can use an internally-generated server ID because the Security WebSphere Common Configuration Model (WCCM) model contains a new tag, internalServerId.

You do not need to specify a server user ID and a password during security configuration except in a mixed-cell environment. See “Administrative roles and naming service authorization” on page 566 for more detailed information about the new internal server ID.

Windows Consider the following issues:

- The server ID needs to be different from the Windows machine name where the product is installed. For example, if the Windows machine name is `vicky` and the security server ID is `vickyy`, the Windows system fails when getting the information (group information, for example) for user `vicky`.
- WebSphere Application Server dynamically determines whether the machine is a member of a Windows system domain.
- WebSphere Application Server does not support Windows trusted domains.
- If a machine is a member of a Windows domain, both the domain user registry and the local user registry of the machine participate in authentication and security role mapping.
- If you use a Windows domain user ID to install and run WebSphere Application Server, the ID must have the following privileges:
  - Be a member of the domain administrative groups in the domain controller
  - Have the Act as part of the operating system privilege in the domain security policy on the domain controller.
  - Have the Act as part of the operating system privilege in the local security policy on the local machine.
  - Have the Log on as a service privilege on the local machine if the server runs as a service.
- The domain user registry takes precedence over the local user registry of the machine and can have undesirable implications if users with the same password exist in both user registries.
- The user that the product processes run under requires the `Administrative` and `Act as part of the operating system` privileges to call the Windows operating system APIs that authenticate or collect user and group information. The process needs special authority, which is given by these privileges. The user in this example might not be the same as the security server ID (the requirement for which is a valid user in the registry). This user logs into the machine (if using the command line to start the product process) or the Log On User setting in the services panel if the product processes have started using the services. If the machine is also part of a domain, this user is a part of the Domain Admin group in the domain to call the operating system APIs in the domain in addition to having the Act as part of operating system privilege in the local machine.

Consider the following points:

- AIX AIX HP-UX Solaris Solaris The user that the product processes run under requires the `root` privilege. This privilege is needed to call the operating system APIs to authenticate or to collect user and group information. The process needs special authority, which is given by the `root` privilege. This user might not be the same as the security server ID (the requirement is that it should be a valid user in the registry). This user logs into the machine and is running the product processes.
- AIX HP-UX Solaris The user that enables administrative security must have the `root` privilege if you use the local operating system registry. Otherwise, a failed validation error is displayed.
- Linux You might need to have the password shadow file in your system.

About this task

The following steps are needed to perform this task initially when setting up security for the first time.

Procedure

1. Click **Security > Global security**.
2. Under User account repository, select **Local operating system** and click **Configure**.
3. Enter a valid user name in the **Primary administrative user name** field. This value is the name of a user with administrative privileges that is defined in the registry. This user name is used to access the administrative console or used by wsadmin.
4. Click **Apply**.

5. Select either the **Automatically generated server identity** or **Server identity that is stored in the repository** option. If you select the **Server identity that is stored in the repository** option, enter the following information:

   Server user ID or administrative user on a Version 6.0.x node
   Specify the short name of the account that is chosen in the second step.

   Server user password
   Specify the password of the account that is chosen in the second step.

6. Click **OK**.

   The administrative console does not validate the user ID and password when you click **OK**. Validation is only done when you click **OK** or **Apply** in the Global security panel. First, make sure that you select **Local operating system** as the available realm definition in the User account repository section, and click **Set as current**. If security was already enabled and you had changed either the user or the password information in this panel, make sure to go to the Global security panel and click **OK** or **Apply** to validate your changes. If your changes are not validated, the server might not start.

   Important: Until you authorize other users to perform administrative functions, you can only access the administrative console with the server user ID and password that you specified. For more information, see "Authorizing access to administrative roles" on page 632.

Results

For any changes in this panel to be effective, you need to save, stop, and start all the product servers, including nodes and application servers. If the server comes up without any problems, the setup is correct.

After completed these steps, you have configured WebSphere Application Server to use the local operating system registry to identify authorized users.

What to do next

Complete any remaining steps for enabling security. For more information, see "Enabling security" on page 69.

Local operating system registries

With the registry implementation for the local operating system, the WebSphere Application Server authentication mechanism can use the user accounts database of the local operating system.

Note: This topic references one or more of the application server log files. As a recommended alternative, you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log` , `SystemErr.log`, `trace.log`, and `activity.log` files on distributed and IBM i systems. You can also use HPEL in conjunction with your native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

Lightweight Directory Access Protocol (LDAP) is a centralized registry. Most local operating system registries are not centralized registries.

WebSphere Application Server provides implementations for the Windows local accounts registry and domain registry, as well as implementations for the Linux, Solaris, and AIX user accounts registries. Windows Active Directory is supported through the LDAP user registry implementation discussed later.

Note: For an Active Directory (domain controller), the three group scopes are Domain Local Group, Global Group, and Universal Group. For an Active Directory (Domain Controller), the two group types are Security and Distribution.

When a group is created, the default value is Global and the default type is Security. With Windows NT

domain registry support for Windows 2003 domain controllers, WebSphere Application Server only supports Global groups that are the Security type. It is recommended that you use the Active Directory registry support rather than a Windows NT domain registry if you use Windows 2003 domain controllers because the Active Directory supports all group scopes and types. The Active Directory also supports a nested group that is not support by Windows NT domain registry. The Active Directory is a centralized control registry.

Note: WebSphere Application Server does not have to install the member of the domain because it can be installed on any machine on any platform. Note that the Windows NT domain native call returns the support group only without an error.

Do not use a local operating system registry in a WebSphere Application Server environment where application servers are dispersed across more than one machine because each machine has its own user registry.

The Windows domain registry and Network Information Services (NIS) are exceptions. Both the Windows domain registry and Network Information Services (NIS) are centralized registries. The Windows domain registry is supported by WebSphere Application Server; however, NIS is not supported.

As mentioned previously, the access IDs taken from the user registry are used during authorization checks. Because these IDs are typically unique identifiers, they vary from machine to machine, even if the exact users and passwords exist on each machine.

Web client certificate authentication is not currently supported when using the local operating system user registry. However, Java client certificate authentication does function with a local operating user registry. Java client certificate authentication maps the first attribute of the certificate domain name to the user ID in the user registry.

Even though Java client certificates function correctly, the following error displays in the `SystemOut.log` file:

CWSCJ0337E: The mapCertificate method is not supported

The error is intended for web client certificates; however, it also displays for Java client certificates. Ignore this error for Java client certificates.

Required privileges

The user that is running the WebSphere Application Server process requires enough operating system privilege to call the Windows systems application programming interface (API) for authenticating and obtaining user and group information from the Windows operating system. This user logs into the machine, or if running as a service, is the Log On As user. Depending on the machine and whether the machine is a stand-alone machine or a machine that is part of a domain or is the domain controller, the access requirements vary.

- For a stand-alone machine, the user:
  - Is a member of the administrative group.
  - Has the Act as part of the operating system privilege.
  - Has the Log on as a service privilege, if the server is run as a service.
- For a machine that is a member of a domain, only a domain user can start the server process and:
  - Is a member of the domain administrative groups in the domain controller.
  - Has the Act as part of the operating system privilege in the Domain security policy on the domain controller.
  - Has the Act as part of the operating system privilege in the Local security policy on the local machine.
  - Has the Log on as a service privilege on the local machine, if the server is run as a service.

    The user is a domain user and not a local user, which implies that when a machine is part of a domain, only a domain user can start the server.

- For a domain controller machine, the user:
  - Is a member of the domain administrative groups in the domain controller.
  - Has the Act as part of the operating system privilege in the Domain security policy on the domain controller.
  - Has the Log on as a service privilege on the domain controller, if the server is run as a service.

If the user running the server does not have the required privilege, you might see one of the following exception messages in the log files:
- A required privilege is not held by the client.
- Access is denied.

Note: Vista 2008 Windows 7 The application server must be started with Administrator privileges if you are using a command prompt. Start the application server from a command prompt window that is launched by performing the following actions:

- Right-click a command prompt shortcut.
- Click **Run As Administrator**.

  When you open the command prompt window as Administrator, an operating-system dialog appears that asks you if you want to continue. Click **Continue** to proceed.

Domain and local user registries

When WebSphere Application Server is started, the security run-time initialization process dynamically attempts to determine if the local machine is a member of a Windows domain. If the machine is part of a domain then by default both the local registry users or groups and the domain registry users or groups can be used for authentication and authorization purposes with the domain registry taking precedence. The list of users and groups that is presented during the security role mapping includes users and groups from both the local user registry and the domain user registry. The users and groups can be distinguished by the associated host names.

WebSphere Application Server does not support trusted domains.

If the machine is not a member of a Windows system domain, the user registry local to that machine is used.

Using both the domain user registry and the local operating system registry

When the machine that hosts the WebSphere Application Server process is a member of a domain, both the local and the domain user registries are used by default. The following section describes more on this topic and recommends some best practices to avoid unfavorable consequences.

Note: Although this section does not directly describe z/OS considerations, you should be aware that overall security operations are affected by how well you set up these registries.

- Best practices
  - In general, if the local and the domain registries do not contain common users or groups, it is simpler to administer and it eliminates unfavorable side effects. If possible, give users and groups access to unique security roles, including the server ID and administrative roles. In this situation, select the users and groups from either the local user registry or the domain user registry to map to the roles.
  - In cases where the same users or groups exist in both the local user registry and the domain user registry, it is recommended that at least the server ID and the users and groups that are mapped to the administrative roles be unique in the registries and exist only in the domain.
  - If a common set of users exists, set a different password to make sure that the appropriate user is authenticated.
- How it works
  - When a machine is part of a domain, the domain user registry takes precedence over the local user registry. For example, when a user logs into the system, the domain user registry tries to

authenticate the user first. If authentication fails, the local user registry is used. When a user or a group is mapped to a role, the user and group information is first obtained from the domain user registry. In case of failure, the local user registry is tried.
  - However, when a fully qualified user or a group name, one with an attached domain or host name, is mapped to a role, only that user registry is used to get the information. Use the administrative console or scripts to get the fully qualified user and group names, which is the recommended way to map users and groups to roles.

    Tip: A user, Bob, on one machine in the local OS user registry, for example, is not the same as the user Bob on another machine in the domain user registry, for example, because the unique ID of Bob, which is the security identifier [SID] in this case, is different in different user registries.
- Examples

  The `MyMachine` machine is part of the `MyDomain` domain. The `MyMachine` machine contains the following users and groups:
  - MyMachine\user2
  - MyMachine\user3
  - MyMachine\group2

  The `MyDomain` domain contains the following users and groups:
  - MyDomain\user1
  - MyDomain\user2
  - MyDomain\group1
  - MyDomain\group2

  Here are some scenarios that assume the previous set of users and groups:
  1. When **user2** logs into the system, the domain user registry is used for authentication. If the authentication fails because the password is different, for example, the local user registry is used.
  2. If the **MyMachine\user2** user is mapped to a role, only the **user2** user in **MyMachine** machine has access. Thus, if the **user2** password is the same on both the local and the domain user registries, the **user2** user cannot access the resource because the **user2** user is always authenticated using the domain user registry. If both user registries have common users, it is recommended that you have different passwords.
  3. If the **group2** group is mapped to a role, only the users who are members of the **MyDomain\group2** group can access the resource because **group2** information is first obtained from the domain user registry.
  4. If the **MyMachine\group2** group is mapped to a role, only the users who are members of the **MyMachine\group2** group can access the resource. A specific group is mapped to the role (**MyMachine\group2** instead of just **group2**).
  5. Use either the **user3** user or the **MyMachine\user3** user to map to a role because the **user3** user is unique as it exists in one user registry only.

  Authorizing with the domain user registry first can cause problems if a user exists in both the domain and local user registries with the same password. Role-based authorization can fail in this situation because the user is first authenticated within the domain user registry. This authentication produces a unique domain security ID that is used in WebSphere Application Server during the authorization check. However, the local user registry is used for role assignment. The domain security ID does not match the unique security ID that is associated with the role. To avoid this problem, map security roles to domain users instead of local users.

Using either the local or the domain user registry

If you want to access users and groups from either the local or the domain user registry, instead of both, set the com.ibm.websphere.registry.UseRegistry property. This property can be set to either `local` or `domain`. When this property is set to `local` (case insensitive) only the local user registry is used. When this property is set to domain, (case insensitive) only the domain user registry is used.

Set this property by completing the following steps to access the **Custom Properties** panel in the administrative console:

1. Click **Security** > **Global security**
2. Under User account repository, click the **Available realm definitions** drop-down list, select **Local operating system**, and click **Configure**.
3. Under Additional properties, click **Custom properties**.

You can also use wsadmin to configure this property. When the property is set, the privilege requirement for the user who is running the product process does not change. For example, if this property is set to `local`, the user that is running the process requires the same privilege, as if the property was not set.

Using system user registries

The following notes apply when you use system user registries:

- AIX HP-UX Linux Solaris When using system user registries, the process ID that runs the WebSphere Application Server process needs the root authority to call the local operating system APIs for authentication and for obtaining user or group information.
- AIX HP-UX Linux Solaris Only the local machine user registry is used. Network Information Service (NIS) (Yellow Pages) is not supported.
- HP-UX If you are using the local operating system user registry, HP-UX must be configured in untrusted mode. Trusted mode is not supported if using the local operating system user registry.
- Linux Solaris

  For WebSphere Application Server local operating system registry to work on the Linux and Solaris platforms, a shadow password file must exist. The shadow password file is named `shadow` and is located in the /etc directory. If the shadow password file does not exist, an error occurs after enabling administrative security and configuring the registry as local operating system.

  To create the shadow file, run the **pwconv** command (with no parameters). This command creates an `/etc/shadow` file from the `/etc/passwd` file. After creating the shadow file, you can enable local operating system security successfully.

Configuring user ID for proper privileges for local operating system registries

Use this page to configure a user ID for proper privileges or to log on as a service on the Windows platform.

Windows

Procedure

1. Click **Start** > **Settings** > **Control Panel** > **Administrative Tools** > **Local Security Policy** > **Local Policies** > **User Rights Assignments** > **Act as part of the operating system** (or **Log on as a service**). For a Windows domain controller, replace **Local Security Policy** with **Domain Security Policy** in the first step.

   Note: If the machine is a stand-alone machine and not a member of a domain, you must add a machineName*userID*, where the *userID* is the owner of the process, such as WebSphere Application Server. If you run WebSphere Application Server as a service, you can log on with `localsystem` as the service.
2. If the machine is a member of a domain, add domainName*userID*, where the *userID* is the owner of process (such as WebSphere Application Server). Start WebSphere Application Server as a service with login ID domainName\userID. If WebSphere Application Server is already in service, go to the service and right-click **IBM WebSphere Application Server > properties >Log on** to change the logon ID and password.
3. Add the user name by clicking **Add**.
4. Restart the machine.

What to do next

Note: In all of the previous configurations, the server can be run as a service using `LocalSystem` for the Log On As entry. The `LocalSystem` entry has the required privileges and there is no need to give special privileges to any user. However, because the `LocalSystem` entry has special privileges, make sure that it is appropriate to use in your environment.

Local operating system settings

Use this page to configure local operating system registry settings.

To view this administrative console page, complete the following steps:

1. Click **Security > Global security**.
2. Under User account repository, click the **Available realm definitions** drop-down list, select **Local operating system**.
3. Click **Configure**.

WebSphere Application Server Version 7.0 distinguishes between the user identities for administrators who manage the environment and server identities for authenticating server to server communications. In most cases, server identities are automatically generated and are not stored in a repository.

However, if you are adding a previous version node to the latest version cell and the previous version node used a server identity and password, you must ensure that the server identity and password for the previous version are defined in the repository for this cell. Enter the server user identity and password on this panel.

Primary administrative user name:

Specifies the name of a user with administrative privileges that is defined in your local operating system.

The user name is used to log on to the administrative console when administrative security is enabled..

Attention: In WebSphere Application Server, Version 6.1 and above, a single user identity is required for both administrative access and internal process communication. When migrating to Version 6.1 and above, this identity is used as the server user identity. You need to specify another user for the administrative user identity.

Automatically generated server identity:

Enables the application server to generate the server identity, which is recommended for environments that contain only Version 6.1 or later nodes. Automatically generated server identities are not stored in a user repository.

Information	Value
Default:	Enabled

Server identity that is stored in the repository:

Specifies a user identity in the repository that is used for internal process communication. Cells that contain Version 6.1 or later nodes require a server user identity that is defined in the active user repository.

Information	Value
Default:	Enabled

Server user ID or administrative user on a Version 6.0.x node:

Specifies the user ID that is used to run the application server for security purposes.

Password:

Specifies the password that corresponds to the server ID.

Local operating system wizard settings

Use this security wizard page to configure local operating system registry settings.

To view this security wizard page, complete the following steps:

1. Click **Security > Global security > Security configuration wizard**.
2. Select your protection settings and click **Next**.
3. Select the **Local operating system** option and click **Next**.

Primary administrative user name:

Specifies the name of a user with administrative privileges that is defined in your local operating system.

The user name is used to log on to the administrative console when administrative security is enabled..

Attention: In WebSphere Application Server, Version 6.1 and above, a single user identity is required for both administrative access and internal process communication. When migrating to Version 6.1 and above, this identity is used as the server user identity. You need to specify another user for the administrative user identity.

Configuring Lightweight Directory Access Protocol user registries

To access a user registry using the Lightweight Directory Access Protocol (LDAP), you must know a valid user name (ID) and password, the server host and port of the registry server, the base distinguished name (DN) and, if necessary, the bind DN and the bind password. You can choose any valid user in the user registry that is searchable. You can use any user ID that has the administrative role to log in.

Before you begin

Note: This topic references one or more of the application server log files. As a recommended alternative, you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log` , `SystemErr.log`, `trace.log`, and `activity.log` files on distributed and IBM i systems. You can also use HPEL in conjunction with your native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

There are two different identities that are used for security purposes: the user ID for administrative functions and the server identity. When administrative security is enabled, the user ID and password for administrative functions is authenticated with the registry. If authentication fails, access to the administrative console is not granted or tasks with wsadmin scripts are not completed. It is important to choose an ID and password that do not expire or change often. If this user ID or password need to change in the registry, make sure that the changes are performed when all the application servers are up and running. When changes are to be made in the registry, review the article on “Standalone Lightweight Directory Access Protocol registries” on page 337 (LDAP) before beginning this task.

The server identity is used for internal process communication. As part of this task, you can change the server identity from the default automatically generated ID to a server ID and password from the LDAP repository.

Procedure

1. In the administrative console, click **Security** > **Global security**.
2. Under User account repository, click the **Available realm definitions** drop-down list, select **Standalone LDAP registry**, and click **Configure**.
3. Enter a valid user name in the **Primary administrative user name** field. Typically, the user name is the short name of the user and is defined by the user filter in the Advanced LDAP settings panel.
4. Determine whether to specify the user identity that is used for internal process communication. Cells that contain Version 5.1 or 6.x nodes require a server user identity that is defined in the active user repository. By default, the **Automatically generated server identity** option is enabled, and the application server generates the server identity. However, you can select the **Server identity that is stored in the repository** option to specify both the server identity and its associated password.
5. Select the type of LDAP server to use from the **Type** list. The type of LDAP server determines the default filters that are used by WebSphere Application Server. These default filters change the **Type** field to **Custom**, which indicates that custom filters are used. This action occurs after you click **OK** or **Apply** in the Advanced LDAP settings panel. Choose the **Custom** type from the list and modify the user and group filters to use other LDAP servers, if required.

   IBM Tivoli Directory Server users can choose IBM Tivoli Directory Server as the directory type. Use the IBM Tivoli Directory Server directory type for better performance. For a list of supported LDAP servers, see the Supported hardware, software, and APIs website.

   Attention: IBM SecureWay Directory Server has been renamed to IBM Tivoli Directory Server in WebSphere Application Server version 6.1.
6. Enter the fully qualified host name of the LDAP server in the **Host** field. You can enter either the IP address or domain name system (DNS) name.
7. Enter the LDAP server port number in the **Port** field. The host name and the port number represent the realm for this LDAP server in the WebSphere Application Server cell. So, if servers in different cells are communicating with each other using Lightweight Third Party Authentication (LTPA) tokens, these realms must match exactly in all the cells.

   The default value is `389`. If multiple WebSphere Application Servers are installed and configured to run in the same single sign-on domain, or if the WebSphere Application Server interoperates with a previous version of the WebSphere Application Server, then it is important that the port number match all configurations. For example, if the LDAP port is explicitly specified as 389 in a version 5.*x* configuration, and a WebSphere Application Server at version 6.0.*x* is going to interoperate with the version 5.*x* server, then verify that port `389` is specified explicitly for the version 6.0.*x* server.

   You can set the com.ibm.websphere.security.ldap.logicRealm custom property to change the value of the realm name that is placed in the token. For more information, see the security custom properties topic.
8. Enter the base distinguished name (DN) in the **Base distinguished name** field. The base DN indicates the starting point for searches in this LDAP directory server. For example, for a user with a DN of `cn=John Doe, ou=Rochester, o=IBM, c=US`, specify the base DN as any of the following options, assuming a suffix of `c=us`:
   - `ou=Rochester, o=IBM, c=us`
   - `o=IBM, c=us`
   - `c=us`

   For authorization purposes, this field is case sensitive by default. Match the case in your directory server. If a token is received (for example, from another cell or Lotus Domino) the base DN in the server must match exactly the base DN from the other cell or Domino. If case sensitivity is not a consideration for authorization, enable the **Ignore case for authorization** option.

   In WebSphere Application Server, the distinguished name is normalized according to the Lightweight Directory Access Protocol (LDAP) specification. Normalization consists of removing spaces in the base distinguished name before or after commas and equal symbols. An example of a

non-normalized base distinguished name is `o = ibm, c = us` or `o=ibm, c=us`. An example of a normalized base distinguished name is `o=ibm,c=us`.

To interoperate between WebSphere Application Server Version 6.0 and later versions, you must enter a normalized base distinguished name in the **Base Distinguished Name** field. In WebSphere Application Server, Version 6.0 or later, the normalization occurs automatically during runtime.

This field is required for all LDAP directories except the Lotus Domino Directory. The **Base Distinguished Name** field is optional for the Domino server.

9. Optional: Enter the bind DN name in the **Bind distinguished name** field. The bind DN is required if anonymous binds are not possible on the LDAP server to obtain user and group information. If the LDAP server is set up to use anonymous binds, leave this field blank. If a name is not specified, the application server binds anonymously. See the **Base Distinguished Name** field description for examples of distinguished names.
10. Optional: Enter the password corresponding to the bind DN in the **Bind password** field.
11. Optional: Modify the Search time out value. This timeout value is the maximum amount of time that the LDAP server waits to send a response to the product client before stopping the request. The default is 120 seconds.
12. Ensure that the **Reuse connection** option is selected. This option specifies that the server should reuse the LDAP connection. Clear this option only in rare situations where a router is used to send requests to multiple LDAP servers and when the router does not support affinity. Leave this option selected for all other situations.
13. Optional: Verify that the **Ignore case for authorization** option is enabled. When you enable this option, the authorization check is case insensitive. Normally, an authorization check involves checking the complete DN of a user, which is unique in the LDAP server and is case sensitive. However, when you use either the IBM Directory Server or the Sun ONE (formerly iPlanet) Directory Server LDAP servers, you must enable this option because the group information that is obtained from the LDAP servers is not consistent in case. This inconsistency affects the authorization check only. Otherwise, this field is optional and can be enabled when a case sensitive authorization check is required. For example, you might select this option when you use certificates and the certificate contents do not match the case of the entry in the LDAP server.

    You can also enable the **Ignore case for authorization** option when you are using single sign-on (SSO) between the product and Lotus Domino. The default is enabled.
14. Optional: Select the **SSL enabled** option if you want to use Secure Sockets Layer communications with the LDAP server.

    Important: This step will only be successful provided that the Signer certificate for the LDAP is first added to the truststore that will be eventually used. If the Signer certificate from the LDAP is not added to the truststore, then

    - An error will be issued by the Administrative console.
    - the deployment manager (DMGR) systemout.log will show the **CWPKI0022E: SSL HANDSHAKE FAILURE** message indicating that the Signer certificate needs to be added to the truststore.

    To ensure an error free operation for this step, You need to first extract to a file the Signer certificate of the LDAP and send that file to the WebSphere Application Server machine. You can then add the certificate to the truststore being defined for the LDAP. In this way, you are assured that the remaining actions for this step will be successful.

    If you select the **SSL enabled** option, you can select either the **Centrally managed** or the **Use specific SSL alias** option.

    Centrally managed
    Enables you to specify an SSL configuration for particular scope such as the cell, node, server, or cluster in one location. To use the **Centrally managed** option, you must specify the SSL configuration for the particular set of endpoints. The Manage endpoint security

configurations and trust zones panel displays all of the inbound and outbound endpoints that use the SSL protocol. If you expand the Inbound or Outbound section of the panel and click the name of a node, you can specify an SSL configuration that is used for every endpoint on that node. For an LDAP registry, you can override the inherited SSL configuration by specifying an SSL configuration for LDAP. To specify an SSL configuration for LDAP, complete the following steps:

a. Click **Security > SSL certificate and key management > Manage endpoint security configurations and trust zones**.
b. Expand **Outbound >** *cell_name* **> Nodes >** *node_name* **> Servers >** *server_name* **> LDAP**.

Use specific SSL alias

Select the **Use specific SSL alias** option if you intend to select one of the SSL configurations in the menu below the option.

This configuration is used only when SSL is enabled for LDAP. The default is *DefaultSSLSettings*. You can click the name of an existing configuration to modify it or complete the following steps to create a new SSL configuration:

a. Click **Security > SSL certificate and key management**.
b. Under Configuration settings, click **Manage endpoint security configurations**.
c. Select a Secure Sockets Layer (SSL) *configuration_name* for selected scopes, such as a cell, node, server, or cluster.
d. Under Related items, click **SSL configurations**.
e. Click **New**.

15. Click **OK** and either **Apply** or **Save** until you return to the Global security panel.

Results

This set of steps is required to set up the LDAP user registry. This step is required as part of enabling security in the WebSphere Application Server.

What to do next

1. If you are enabling security, complete the remaining steps as specified in “Enabling security for the realm” on page 84.
2. Save, stop, and restart all the product servers (deployment managers, nodes and Application Servers) for changes in this panel to take effect. If the server comes up without any problems the setup is correct.

Standalone LDAP registry settings

Use this page to configure Lightweight Directory Access Protocol (LDAP) settings when users and groups reside in an external LDAP directory.

To view this administrative console page, complete the following steps:

1. Click **Security** > **Global security**.
2. Under User account repository, click the `Available realm definitions` drop-down list, select `Standalone LDAP registry`, and click `Configure`.

When security is enabled and any of these properties change, go to the Global security panel and click `Apply` to validate the changes.

WebSphere Application Server Version 7.0 distinguishes between the user identities for administrators who manage the environment and server identities for authenticating server to server communications. In most cases, server identities are automatically generated and are not stored in a repository.

However, if you are adding a previous version node to the latest version cell and the previous version node used a server identity and password, you must ensure that the server identity and password for the previous version are defined in the repository for this cell. Enter the server user identity and password on this panel.

Note: The initial profile creation configures WebSphere Application Server to use a federated repositories security registry option with the file-based registry. This security registry configuration can be changed to use other options, including the stand-alone LDAP registry. Instead of changing from the federated repositories option to the stand-alone LDAP registry option under the User account repository configuration, consider employing the federated repositories option, which provides for LDAP configuration. Federated repositories provide a wide range of capabilities, including the ability to have one or multiple user registries. It supports federating one or more LDAPs in addition to file-based and custom registries. It also has improved failover capabilities, and a robust set of member (user and group) management capabilities. Federated repositories is required when you are using the new member management capabilities in WebSphere Portal 6.1 and above, and Process Server 6.1 and above. The use of federated repositories is required for following LDAP referrals, which is a common requirement in some LDAP server environments (such as Microsoft Active Directory).

It is recommended that you migrate from stand-alone LDAP registries to federated repositories. If you move to WebSphere Portal 6.1 and above, and or WebSphere Process Server 6.1 and above, you should migrate to federated repositories prior to these upgrades. For more information about federated repositories and its capabilities, read the Federated repositories topic. For more information about how to migrate to federated repositories, read the Migrating a stand-alone LDAP repository to a federated repositories LDAP repository configuration topic.

Primary administrative user name:

Specifies the name of a user with administrative privileges that is defined in your user registry.

The user name is used to log onto the administrative console when administrative security is enabled. Versions 6.1 and later require an administrative user that is distinct from the server user identity so that administrative actions can be audited.

Attention: In WebSphere Application Server, Version 6.x, a single user identity is required for both administrative access and internal process communication. When you migrate to Version 8.x, this identity is used as the server user identity. You need to specify another user for the administrative user identity.

Automatically generated server identity:

Enables the application server to generate the server identity, which is recommended for environments that contain only Version 6.1 or later nodes. Automatically generated server identities are not stored in a user repository.

Information	Value
Default:	Enabled

Server identity that is stored in the repository:

Specifies a user identity in the repository that is used for internal process communication. Cells that contain Version 6.1 or later nodes require a server user identity that is defined in the active user repository.

Information	Value
Default:	Enabled

Server user ID or administrative user on a Version 6.0.x node:

Specifies the user ID that is used to run the application server for security purposes.

Password:

Specifies the password that corresponds to the server ID.

Type of LDAP server:

Specifies the type of LDAP server to which you connect.

IBM SecureWay Directory Server is not supported.

Host:

Specifies the host ID (IP address or domain name service (DNS) name) of the LDAP server.

Port:

Specifies the host port of the LDAP server.

If multiple application servers are installed and configured to run in the same single sign-on domain or if the application server interoperates with a previous version, it is important that the port number match all configurations. For example, if the LDAP port is explicitly specified as `389` in a Version 6.1 and above configuration, and a WebSphere Application Server at Version 8.x is going to interoperate with the Version 6.1 and above server, verify that port `389` is specified explicitly for the Version 8.x server.

Information	Value
Default:	389
Type:	Integer

Base distinguished name (DN):

Specifies the base distinguished name (DN) of the directory service, which indicates the starting point for LDAP searches of the directory service. In most cases, bind DN and bind password are needed. However, when anonymous bind can satisfy all of the required functions, bind DN and bind password are not needed.

For example, for a user with a DN of `cn=John Doe , ou=Rochester, o=IBM, c=US`, specify the Base DN as any of the following options: `ou=Rochester, o=IBM, c=US` or `o=IBM c=US` or `c=US`. For authorization purposes, this field is case sensitive. This specification implies that if a token is received, for example, from another cell or Lotus Domino, the base DN in the server must match the base DN from the other cell or Lotus Domino server exactly. If case sensitivity is not a consideration for authorization, enable the `Ignore case for authorization` option. This option is required for all Lightweight Directory Access Protocol (LDAP) directories, except for the Lotus Domino Directory, IBM Tivoli Directory Server V6.0, and Novell eDirectory, where this field is optional.

Bind distinguished name (DN):

Specifies the DN for the application server to use when binding to the directory service.

If no name is specified, the application server binds anonymously. See the Base distinguished name (DN) field description for examples of distinguished names.

Bind password:

Specifies the password for the application server to use when binding to the directory service.

Search timeout:

Specifies the timeout value in seconds for a Lightweight Directory Access Protocol (LDAP) server to respond before stopping a request.

Information	Value
Default:	120

Reuse connection:

Specifies whether the server reuses the LDAP connection. Clear this option only in rare situations where a router is used to distribute requests to multiple LDAP servers and when the router does not support affinity.

Information	Value
Default:	Enabled
Range:	Enabled or Disabled

Important: Disabling the `Reuse connection` option causes the application server to create a new LDAP connection for every LDAP search request. This situation impacts system performance if your environment requires extensive LDAP calls. This option is provided because the router is not sending the request to the same LDAP server. The option is also used when the idle connection timeout value or firewall timeout value between the application server and LDAP is too small.

If you are using WebSphere Edge Server for LDAP failover, you must enable TCP resets with the Edge server. A TCP reset causes the connection to immediately closed and a backup server to failover. For more information, see “Sending TCP resets when server is down” at http://www.ibm.com/software/webservers/appserv/doc/v50/ec/infocenter/edge/LBguide.htm#HDRRESETSERVER and the Edge Server V2 - TCP Reset feature in PTF #2 described in: http://publibfp.dhe.ibm.com/epubs/pdf/i1032540.pdf.

Ignore case for authorization:

Specifies that a case insensitive authorization check is performed when using the default authorization.

This option is required when IBM Tivoli Directory Server is selected as the LDAP directory server.

This option is required when Sun ONE Directory Server is selected as the LDAP directory server. For more information, see “Using specific directory servers as the LDAP server” in the documentation.

This option is optional and can be enabled when a case-sensitive authorization check is required. For example, use this option when the certificates and the certificate contents do not match the case that is used for the entry in the LDAP server. You can enable the `Ignore case for authorization` option when using single sign-on (SSO) between the application server and Lotus Domino.

Information	Value
Default:	Enabled
Range:	Enabled or Disabled

SSL enabled:

Specifies whether secure socket communication is enabled to the Lightweight Directory Access Protocol (LDAP) server.

When enabled, the LDAP Secure Sockets Layer (SSL) settings are used, if specified.

Centrally managed:

Specifies that the selection of an SSL configuration is based upon the outbound topology view for the Java Naming and Directory Interface (JNDI) platform.

Centrally managed configurations support one location to maintain SSL configurations rather than spreading them across the configuration documents.

Information	Value
Default:	Enabled

Use specific SSL alias:

Specifies the SSL configuration alias to use for LDAP outbound SSL communications.

This option overrides the centrally managed configuration for the JNDI platform.

Standalone LDAP registry wizard settings

Use this security wizard page to provide the basic settings to connect the application server to an existing Lightweight Directory Access Protocol (LDAP) registry.

To view this security wizard page, click **Security > Global security > Security configuration wizard**. You can modify your LDAP registry configuration by completing the following steps:

1. Click **Security > Global security**.
2. Under User account repository, click the **Available realm definitions** drop-down list, select**Standalone LDAP registry**, and click **Configure**.

Primary administrative user name:

Specifies the name of a user with administrative privileges that is defined in your user registry.

The user name is used to log onto the administrative console when administrative security is enabled. Versions 6.1 and later require an administrative user that is distinct from the server user identity so that administrative actions can be audited.

Attention: In WebSphere Application Server, Version 6.x, a single user identity is required for both administrative access and internal process communication. When you migrate to Version 8.x, this identity is used as the server user identity. You need to specify another user for the administrative user identity.

Type of LDAP server:

Specifies the type of LDAP server to which you connect.

IBM SecureWay Directory Server is not supported.

Host:

Specifies the host ID (IP address or domain name service (DNS) name) of the LDAP server.

Port:

Specifies the host port of the LDAP server.

If multiple application servers are installed and configured to run in the same single sign-on domain or if the application server interoperates with a previous version, it is important that the port number match all configurations. For example, if the LDAP port is explicitly specified as `389` in a Version 6.1 and above configuration, and a WebSphere Application Server at Version 8.x is going to interoperate with the Version 6.1 and above server, verify that port `389` is specified explicitly for the Version 8.x server.

Information	Value
Default:	389
Type:	Integer

Base distinguished name (DN):

Specifies the base distinguished name (DN) of the directory service, which indicates the starting point for LDAP searches of the directory service. In most cases, bind DN and bind password are needed. However, when anonymous bind can satisfy all of the required functions, bind DN and bind password are not needed.

For example, for a user with a DN of `cn=John Doe , ou=Rochester, o=IBM, c=US`, specify the Base DN as any of the following options: `ou=Rochester, o=IBM, c=US` or `o=IBM, c=US` or `c=US`. For authorization purposes, this field is case sensitive. This specification implies that if a token is received, for example, from another cell or Lotus Domino, the base DN in the server must match the base DN from the other cell or Lotus Domino server exactly.

Bind distinguished name (DN):

Specifies the DN for the application server to use when binding to the directory service.

If no name is specified, the application server binds anonymously. See the Base distinguished name (DN) field description for examples of distinguished names.

Bind password:

Specifies the password for the application server to use when binding to the directory service.

Advanced Lightweight Directory Access Protocol user registry settings

Use this page to configure the advanced Lightweight Directory Access Protocol (LDAP) user registry settings when users and groups reside in an external LDAP directory.

To view this administrative page, complete the following steps:

1. Click **Security** > **Global security**.
2. Under User account repository, click the **Available realm definitions** drop-down list, select **Standalone LDAP registry**, and click **Configure**.
3. Under Additional properties, click **Advanced Lightweight Directory Access Protocol (LDAP) user registry settings**.

Default values for all the user and group related filters are already completed in the appropriate fields. You can change these values depending on your requirements. These default values are based on the type of LDAP server that is selected in the Standalone LDAP registry settings panel. If this type changes, for example from Netscape to Secureway, the default filters automatically change. When the default filter

values change, the LDAP server type changes to `Custom` to indicate that custom filters are used. When security is enabled and any of these properties change, go to the Global security panel and click **Apply** or **OK** to validate the changes.

Note: The initial profile creation configures WebSphere Application Server to use a federated repositories security registry option with the file-based registry. This security registry configuration can be changed to use other options, including the stand-alone LDAP registry. Instead of changing from the federated repositories option to the stand-alone LDAP registry option under the User account repository configuration, consider employing the federated repositories option, which provides for LDAP configuration. Federated repositories provide a wide range of capabilities, including the ability to have one or multiple user registries. It supports federating one or more LDAPs in addition to file-based and custom registries. It also has improved failover capabilities, and a robust set of member (user and group) management capabilities. Federated repositories is required when you are using the new member management capabilities in WebSphere Portal 6.1 and above, and Process Server 6.1 and above. The use of federated repositories is required for following LDAP referrals, which is a common requirement in some LDAP server environments (such as Microsoft Active Directory).

It is recommended that you migrate from stand-alone LDAP registries to federated repositories. If you move to WebSphere Portal 6.1 and above, and or WebSphere Process Server 6.1 and above, you should migrate to federated repositories prior to these upgrades. For more information about federated repositories and its capabilities, read the Federated repositories topic. For more information about how to migrate to federated repositories, read the Migrating a stand-alone LDAP repository to a federated repositories LDAP repository configuration topic.

User filter:

Specifies the LDAP user filter that searches the user registry for users.

This option is typically used for security role-to-user assignments and specifies the property by which to look up users in the directory service. For example, to look up users based on their user IDs, specify `(&(uid=%v)(objectclass=inetOrgPerson))`. For more information about this syntax, see the LDAP directory service documentation.

Information	**Value**
Data type:	String

Group filter:

Specifies the LDAP group filter that searches the user registry for groups

This option is typically used for security role-to-group assignments and specifies the property by which to look up groups in the directory service. For more information about this syntax, see the LDAP directory service documentation.

Information	**Value**
Data type:	String

User ID map:

Specifies the LDAP filter that maps the short name of a user to an LDAP entry.

Specifies the piece of information that represents users when users display. For example, to display entries of the `object class = inetOrgPerson` type by their IDs, specify `inetOrgPerson:uid`. This field takes multiple objectclass:property pairs delimited by a semicolon (;).

Information	Value
Data type:	String

Group ID map:

Specifies the LDAP filter that maps the short name of a group to an LDAP entry.

Specifies the piece of information that represents groups when groups display. For example, to display groups by their names, specify `*:cn`. The asterisk (`*`) is a wildcard character that searches on any object class in this case. This field takes multiple `objectclass:property` pairs, delimited by a semicolon (;).

Information	Value
Data type:	String

Group member ID map:

Specifies the LDAP filter that identifies user-to-group relationships.

For directory types SecureWay, and Domino, this field takes multiple objectclass:property pairs, delimited by a semicolon (;). In an objectclass:property pair, the object class value is the same object class that is defined in the group filter, and the property is the member attribute. If the object class value does not match the object class in the group filter, authorization might fail if groups are mapped to security roles. For more information about this syntax, see your LDAP directory service documentation.

For IBM Directory Server, Sun ONE, and Active Directory, this field takes multiple `group attribute:member attribute` pairs delimited by a semicolon (;). These pairs are used to find the group memberships of a user by enumerating all the group attributes that are possessed by a given user. For example, attribute pair `memberof:member` is used by Active Directory, and ibm-allGroup:member is used by IBM Directory Server. This field also specifies which property of an object class stores the list of members belonging to the group represented by the object class. For supported LDAP directory servers, see “Supported directory services”.

Information	Value
Data type:	String

Perform a nested group search:

Specifies a recursive nested group search.

Select this option if the Lightweight Directory Access Protocol (LDAP) server does not support recursive server-side group member searches and if recursive group member search is required. It is not recommended that you select this option to locate recursive group memberships for LDAP servers. Application server security leverages the recursive search functionality of the LDAP server to search a user's group memberships, including recursive group memberships. For example:

- IBM Directory Server is preconfigured by the application server security to recursively calculate a user's group memberships using the `ibm-allGroup` attribute.
- SunONE directory server is preconfigured to calculate nested group memberships using the `nsRole` attribute.

Information	Value
Data type:	String

Kerberos user filter:

Specifies the Kerberos user filter value. This value can be modified when Kerberos is configured and is active as one of the preferred authentication mechanisms.

Information	Value
Data type:	String

Certificate map mode:

Specifies whether to map X.509 certificates into an LDAP directory by EXACT_DN or CERTIFICATE_FILTER. Specify CERTIFICATE_FILTER to use the specified certificate filter for the mapping.

Information	Value
Data type:	String

Certificate filter:

Specifies the filter certificate mapping property for the LDAP filter. The filter is used to map attributes in the client certificate to entries in the LDAP registry.

If more than one LDAP entry matches the filter specification at runtime, authentication fails because the result is an ambiguous match. The syntax or structure of this filter is: `(&(uid=${SubjectCN})(objectclass=inetOrgPerson))`. The left side of the filter specification is an LDAP attribute that depends on the schema that your LDAP server is configured to use. The right side of the filter specification is one of the public attributes in your client certificate. The right side must begin with a dollar sign ($) and open bracket ({) and end with a close bracket (}). You can use the following certificate attribute values on the right side of the filter specification. The case of the strings is important:

- `${UniqueKey}`
- `${PublicKey}`
- `${IssuerDN}`
- `${Issuer<xx>}`

  where *<xx>* is replaced by the characters that represent any valid component of the Issuer Distinguished Name. For example, you might use `${IssuerCN}` for the Issuer Common Name.
- `${NotAfter}`
- `${NotBefore}`
- `${SerialNumber}`
- `${SigAlgName}`
- `${SigAlgOID}`
- `${SigAlgParams}`
- `${SubjectDN}`
- `${Subject<xx>}`

  where *<xx>* is replaced by the characters that represent any valid component of the Subject Distinguished Name. For example, you might use `${SubjectCN}` for the Subject Common Name.
- `${Version}`

Information	Value
Data type:	String

Configuring Lightweight Directory Access Protocol search filters

Use this topic to configure the LDAP search filters. These steps are required to modify existing user and group filters for a particular LDAP directory type, and also to set up certificate filters to map certificates to entries in the LDAP server.

Before you begin

WebSphere Application Server uses Lightweight Directory Access Protocol (LDAP) filters to search and obtain information about users and groups from an LDAP directory server. A default set of filters is provided for each LDAP server that the product supports. You can modify these filters to fit your LDAP configuration. After the filters are modified and you click **OK** or **Apply** the directory type in the Standalone LDAP registry panel changes to `custom`, which indicates that custom filters are used. Also, you can develop filters to support any additional type of LDAP server. The effort to support additional LDAP directories is optional and other LDAP directory types are not supported. Complete the following steps in the administrative console.

Procedure

1. Click **Security > Global security**.
2. Under User account repository, select **Standalone LDAP registry** and click **Configure**.
3. Under Additional properties, click **Advanced Lightweight Directory Access Protocol (LDAP) user registry settings**.
4. Modify the user filter, if necessary. The user filter is used for searching the registry for users and is typically used for the security role-to-user assignment. The filter is also used to authenticate a user with the attribute that is specified in the filter. The filter specifies the property that is used to look up users in the directory service.

   In the following example, the property that is assigned to `%v`, which is the short name of the user, must be a unique key. Two LDAP entries with the same object class cannot have the same short name. To look up users based on their user IDs (uid) and to use the inetOrgPerson object class, specify the following syntax:

   ```
   (&(uid=%v)(objectclass=inetOrgPerson)
   ```

   For more information about this syntax, see the “Using specific directory servers as the LDAP server” on page 185 documentation.
5. Modify the Kerberos user filter, if necessary. The Kerberos user filter name is used for searching the registry for the Kerberos principal name. Specify the LDAP attribute that holds the Kerberos principal name.

   IBM Lotus Domino default krbuser filter:
   (&(krbPrincipalName=%v)(objectcategory=Person))

   IBM SecureWay Directory Server default krbuser filter:
   (&(krbPrincipalName=%v)(objectcategory=ePerson))

   Microsoft Active Directory default krbuser filter:
   (&(userprincipalname=%v)(objectcategory=user))

   Sun Java System Directory Server default krbuser filter:
   (&(krbPrincipalName=%v)(objectcategory=inetOrgPerson))

   Novell eDirectory default krbuser filter:
   (&(krbPrincipalName=%v)(objectcategory=Person))
6. Optional: If your using Federated Repositories, modify the Kerberos attribute name if necessary. The Kerberos attribute name is used for searching the registry for Kerberos principal. Specify the LDAP attribute that holds the Kerberos principal name.

   IBM Lotus Domino default krbuser filter:
   krbPrincipalName

   IBM SecureWay Directory Server default krbuser filter:
   krbPrincipalName

   Microsoft Active Directory default krbuser filter:
   userprincipalname

Sun Java System Directory Server default krbuser filter:
krbPrincipalName

Novell eDirectory default krbuser filter:
krbPrincipalName

7. Modify the group filter, if necessary. The group filter is used in searching the registry for groups and is typically used for the security role-to-group assignment. Also, the filter is used to specify the property by which to look up groups in the directory service.

   In the following example, the property that is assigned to `%v`, which is the short name of the group, must be a unique key. Two LDAP entries with the same object class cannot have the same short name. To look up groups based on their common names (CN) and to use either the groupOfNames object class or the groupOfUniqueNames object class, specify the following syntax:

   ```
   (&(cn=%v)(|(objectclass=groupOfNames)(objectclass=groupOfUniqueNames)))
   ```

   For more information about this syntax, see the “Using specific directory servers as the LDAP server” on page 185 documentation.
8. Modify the user ID map, if necessary. This filter maps the short name of a user to an LDAP entry and specifies the piece of information that represents users when these users are displayed with their short names. For example, to display entries of `object class = inetOrgPerson` by their IDs, specify `inetOrgPerson:uid`. This field takes multiple objectclass:property pairs, delimited by a semicolon (;). To provide a consistent value for methods like the getCallerPrincipal method and the getUserPrincipal method, the short name that is obtained by using this filter is used. For example, the `CN=Bob Smith, ou=austin.ibm.com, o=IBM, c=US` user can log in using any attributes that are defined, for example, email address, social security number, and so on, but when these methods are called, the `bob` user ID is returned no matter how the user logs in.

   Note: Only the getUserDisplayName API honors the user ID map.
9. Modify the group ID map filter, if necessary. This filter maps the short name of a group to an LDAP entry and specifies the piece of information that represents groups when groups display. For example, to display groups by their names, specify `*:cn`. The asterisk (*) is a wildcard character that searches on any object class in this case. This field takes multiple objectclass:property pairs, delimited by a semicolon (;).
10. Modify the group member ID map filter, if necessary. This filter identifies user-to-group memberships. For SecureWay, and Domino directory types, this field is used to query all the groups that match the specified object classes to see if the user is contained in the specified attribute. For example, to get all the users that belong to groups with the groupOfNames object class and the users that are contained in the member attributes, specify `groupOfNames:member`. This syntax, which is a property of an object class, stores the list of members that belong to the group that is represented by the object class. This field takes multiple objectclass:property pairs that are delimited by a semicolon (;). For more information about this syntax, see the “Using specific directory servers as the LDAP server” on page 185.

    For the IBM Tivoli Directory Server, Sun ONE, and Active Directory, this field is used to query all users in a group with the information that is stored in the user object. For example, the memberof:member filter (for Active Directory) is used to get the memberof attribute of the user object to obtain all the groups to which the user belongs. The member attribute is used to get all the users in a group that use the Group object. Using the User object to obtain the group information improves performance.
11. Select the **Perform a nested group search** option if your LDAP server does not support recursive server-side searches.
12. Modify the Certificate map mode, if necessary. You can use the X.590 certificates for user authentication when LDAP is selected as the registry. This field is used to indicate whether to map the X.509 certificates into an LDAP directory user by **EXACT_DN** or **CERTIFICATE_FILTER**. If **EXACT_DN** is selected, the DN in the certificate must exactly match the user entry in the LDAP server, including case and spaces.

Select the **Ignore case for authorization** option on the Standalone LDAP registry settings to make the authorization case insensitive. To access the Standalone LDAP registry settings panel, complete the following steps:

a. Click **Security > Global security**.

b. Under User account repository, click the **Available realm definitions** drop-down list, select**Standalone LDAP registry**.

13. If you select **CERTIFICATE_FILTER**, specify the LDAP filter for mapping attributes in the client certificate to entries in LDAP. If more than one LDAP entry matches the filter specification at run time, authentication fails because an ambiguous match results. The syntax or structure of this filter is: `LDAP attribute=${Client certificate attribute}` (for example, `uid=${SubjectCN}`).

    The left side of the filter specification is an LDAP attribute that depends on the schema that your LDAP server is configured to use. The right side of the filter specification is one of the public attributes in your client certificate. Note that the right side must begin with a dollar sign ($), open bracket ({), and end with a close bracket (}). Use the following certificate attribute values on the right side of the filter specification. The case of the strings is important.

    - `${UniqueKey}`
    - `${PublicKey}`
    - `${IssuerDN}`
    - `${Issuer<xx>}`

      where *<xx>* is replaced by the characters that represent any valid component of the Issuer Distinguished Name. For example, you might use `${IssuerCN}` for the Issuer Common Name.
    - `${NotAfter}`
    - `${NotBefore}`
    - `${SerialNumber}`
    - `${SigAlgName}`
    - `${SigAlgOID}`
    - `${SigAlgParams}`
    - `${SubjectDN}`
    - `${Subject<xx>}`

      where *<xx>* is replaced by the characters that represent any valid component of the Subject Distinguished Name. For example, you might use `${SubjectCN}` for the Subject Common Name.
    - `${Version}`

    To enable this field, select **CERTIFICATE_FILTER** for the certificate mapping.

    gotcha: Subject alternative names (SANs) are not supported as certificate filter items.

14. Click **Apply**.

    When any LDAP user or group filter is modified in the Advanced LDAP Settings panel click **Apply**. Clicking **OK** navigates you to the Standalone LDAP registry panel, which contains the previous LDAP directory type, rather than the custom LDAP directory type. Clicking **OK** or **Apply** in the Standalone LDAP registry panel saves the back-level LDAP directory type and the default filters of that directory. This action overwrites any changes to the filters that you made. To avoid overwriting changes, you can take either of the following actions:

    - Click **Apply** in the Advanced Lightweight Directory Access Protocol (LDAP) user registry settings panel. Click **Security > Global security** and change the User account repository type to `Stand-alone custom registry`.
    - Select **Custom** type from the Standalone LDAP registry panel. Click **Apply** and then change the filters by clicking the Advanced Lightweight Directory Access Protocol (LDAP) user registry settings panel. After you complete your changes, click **Apply** or **OK**.

    The validation of the changes does not take place in this panel. Validation is done when you click **OK** or **Apply** on the Global security panel. If you are in the process of enabling security for the first time, complete the remaining steps and go to the Global security panel. Select **Standalone LDAP registry** as the user account repository. If security is already enabled and any information on this panel

changes, go to the Global security panel and click **OK** or **Apply** to validate your changes. If your changes are not validated, the server might not start.

Results

These steps result in the configuration of the LDAP search filters. These steps are required to modify existing user and group filters for a particular LDAP directory type. The steps are also used to set up certificate filters to map certificates to entries in the LDAP server.

What to do next

1. Validate this setup by clicking **OK** or **Apply** on the Global security panel.
2. Save, stop, and start all the product servers, including the cell, nodes and all of the application servers for any changes in this panel to become effective.
3. After the server starts, go through all the security-related tasks (getting users, getting groups, and so on) to verify that the changes to the filters function.

Using specific directory servers as the LDAP server

This article provides important information about the directory servers that are supported as Lightweight Directory Access Protocol (LDAP) servers in WebSphere Application Server.

Before you begin

Microsoft Active Directory forests are not supported with the stand-alone LDAP Registry. The Federated Repository Registry, when configured to use an Active Directory LDAP does support the use of forests.

About this task

For a list of supported LDAP servers, refer to the Supported hardware and software website.

It is expected that other LDAP servers follow the LDAP specification. Support is limited to these specific directory servers only. You can use any other directory server by using the custom directory type in the list and by filling in the filters that are required for that directory.

To improve performance for LDAP searches, the default filters for IBM Tivoli Directory Server, Sun ONE, and Active Directory are defined such that when you search for a user, the result contains all the relevant information about the user (user ID, groups, and so on). As a result, the product does not call the LDAP server multiple times. This definition is possible only in these directory types, which support searches where the complete user information is obtained.

If you use the IBM Directory Server, select the **Ignore case for authorization** option. This option is required because when the group information is obtained from the user object attributes, the case is not the same as when you get the group information directly. For the authorization to work in this case, perform a case insensitive check and verify the requirement for the **Ignore case for authorization** option.

- **Using IBM Tivoli Directory Server as the LDAP server**

  To use IBM Tivoli Directory Server, formerly IBM Directory Server, select **IBM Tivoli Directory Server** as the directory type.

  The difference between these two types is group membership lookup. It is recommended that you choose the IBM Tivoli Directory Server for optimum performance during runtime. In the IBM Tivoli Directory Server, the group membership is an operational attribute. With this attribute, a group membership lookup is done by enumerating the ibm-allGroups attribute for the entry. All group memberships, including the static groups, dynamic groups, and nested groups, can be returned with the `ibm-allGroups` attribute.

WebSphere Application Server supports dynamic groups, nested groups, and static groups in IBM Tivoli Directory Server using the `ibm-allGroups` attribute. To utilize this attribute in a security authorization application, use a case-insensitive match so that attribute values returned by the ibm-allGroups attribute are all in uppercase.

Important: It is recommended that you do not install IBM Tivoli Directory Server Version 6.0 on the same machine that you install Version 8.5. IBM Tivoli Directory Server Version 6.0 includes WebSphere Application Server, Express Version 5.1.1, which the directory server uses for its administrative console. Install the Web Administration tool Version 6.0 and WebSphere Application Server, ExpressVersion 5.1.1, which are both bundled with IBM Tivoli Directory Server Version 6.0, on a different machine from Version 8.5. You cannot use Version 8.5 as the administrative console for IBM Tivoli Directory Server. If IBM Tivoli Directory Server Version 6.0 and Version 8.5 are installed on the same machine, you might encounter port conflicts.

If you must install IBM Tivoli Directory Server Version 6.0 and Version 8.5 on the same machine, consider the following information:

- During the IBM Tivoli Directory Server installation process, you must select both the **Web Administration tool** and **WebSphere Application Server, Express Version 5.1.1**.
- Install Version 8.5.
- When you install Version 8.5, change the port number for the application server.
- You might need to adjust the WebSphere Application Server environment variables on Version 8.5 for *WAS_HOME* and *WAS_INSTALL_ROOT* (or *APP_SERVER_ROOT* for IBM i). To change the variables using the administrative console, click **Environment > WebSphere Variables**.

- **Using a Lotus Domino Enterprise Server as the LDAP server**

  If you select the Lotus Domino Enterprise Server Version 6.5.4 or Version 7.0 and the attribute short name is not defined in the schema, you can take either of the following actions:
  - Change the schema to add the short name attribute.
  - Change the user ID map filter to replace the short name with any other defined attribute (preferably to UID). For example, change `person:shortname` to `person:uid`.

  The userID map filter is changed to use the uid attribute instead of the shortname attribute as the current version of Lotus Domino does not create the shortname attribute by default. If you want to use the shortname attribute, define the attribute in the schema and change the userID map filter.

  ```
  User ID Map :    person:shortname
  ```

- **Using Sun ONE Directory Server as the LDAP server**

  You can select **Sun ONE Directory Server** for your Sun ONE Directory Server system. In Sun ONE Directory Server, the object class is the default groupOfUniqueName when you create a group. For better performance, WebSphere Application Server uses the User object to locate the user group membership from the nsRole attribute. Create the group from the role. If you want to use the groupOfUniqueName attribute to search groups, specify your own filter setting. Roles unify entries. Roles are designed to be more efficient and easier to use for applications. For example, an application can locate the role of an entry by enumerating all the roles that are possessed by a given entry, rather than selecting a group and browsing through the members list. When using roles, you can create a group using a:
  - Managed role
  - Filtered role
  - Nested role

  All of these roles are computable by the nsRole attribute.

- **Using Microsoft Active Directory server as the LDAP server**

To use Microsoft Active Directory as the LDAP server for authentication with WebSphere Application Server you must take specific steps. By default, Microsoft Active Directory does not permit anonymous LDAP queries. To create LDAP queries or to browse the directory, an LDAP client must bind to the LDAP server using the distinguished name (DN) of an account that has the authority to search and read the values of LDAP attributes, such as user and group information, needed by the Application Server. A group membership search in the Active Directory is done by enumerating the `memberof` attribute for a given user entry, rather than browsing through the member list in each group. If you change the default behavior to browse each group, you can change the **Group Member ID Map** field from `memberof:member` to `group:member`.

The following steps describe how to set up Microsoft Active Directory as your LDAP server.

Procedure

1. Determine the full distinguished name (DN) and password of an account in the administrators group.

   For example, if the Active Directory administrator creates an account in the Users folder of the Active Directory Users and Computers Windows control panel and the DNS domain is `ibm.com`, the resulting DN has the following structure:

```
cn=<adminUsername>, cn=users, dc=ibm,
dc=com
```

2. Determine the short name and password of any account in the Microsoft Active Directory.
3. Use the WebSphere Application Server administrative console to set up the information that is needed to use Microsoft Active Directory.
   a. Click **Security > Global security**.
   b. Under User account repository, select **Standalone LDAP registry** and click **Configure**.
   c. Set up LDAP with Active Directory as the type of LDAP server. Based on the information that is determined in the previous steps, you can specify the following values on the LDAP settings panel:

      Primary administrative user name
      Specify the name of a user with administrative privileges that is defined in the registry. This user name is used to access the administrative console or used by wsadmin.

      Type Specify Active Directory

      Host Specify the domain name service (DNS) name of the machine that is running Microsoft Active Directory.

      Base distinguished name (DN)
      Specify the domain components of the DN of the account that is chosen in the first step. For example: `dc=ibm, dc=com`

      Bind distinguished name (DN)
      Specify the full distinguished name of the account that is chosen in the first step. For example: `cn=adminUsername, cn=users, dc=ibm, dc=com`

      Bind password
      Specify the password of the account that is chosen in the first step.

   d. Click **OK** and **Save** to save the changes to the master configuration.
4. Click **Security > Global security**.
5. Under User account repository, click the **Available realm definitions** drop-down list, select **Standalone LDAP registry**, and click **Configure**.
6. Select either the **Automatically generated server identity** or **Server identity that is stored in the repository** option. If you select the **Server identity that is stored in the repository** option, enter the following information:

   Server user ID or administrative user on a Version 6.0.x node
   Specify the short name of the account that is chosen in the second step.

Server user password
Specify the password of the account that is chosen in the second step.

7. Optional: Set ObjectCategory as the filter in the Group member ID map field to improve LDAP performance.
   a. Under Additional properties, click **Advanced Lightweight Directory Access Protocol (LDAP) user registry settings** .
   b. Add `;objectCategory:group` to the end of the Group member ID map field.
8. Click **OK** and **Save** to save the changes to the master configuration.
9. Stop and restart the administrative server so that the changes take effect.

Locating user group memberships in a Lightweight Directory Access Protocol registry

You can configure WebSphere Application Server security to use Lightweight Directory Access Protocol (LDAP) servers. The LDAP specifications allow for different mechanisms to define group memberships. Depending on your LDAP server implementation, you can use methods to determine group memberships. WebSphere Application Server can search group memberships directly or indirectly. Also, you can configure the product to search one or more static groups, recursive or nested groups, and dynamic groups for some Lightweight Directory Access Protocol (LDAP) servers.

Procedure

- Evaluate group memberships.
  – **Static group membership:** All LDAP server implementations support static group membership. The group object contains a list of users or groups that are members of the group. To determine the groups in which a user is a member, you must get the list of all groups, and then query each group in turn to see if the user is a member of that group. This operation results in (0)zero groups and does not scale well.

    Several LDAP servers enable user objects in the LDAP server to contain information about the groups to which they belong. Examples of LDAP servers that support direct group searches include Microsoft Active Directory Server and the owner of eDirectory.

  – **Dynamic group memberships**

    Some user group memberships are computable from attributes within the user object. IBM Directory Server and Sun ONE Directory Server are two examples of LDAP servers that support dynamic group membership. In some LDAP servers, you can use an attribute to include a user's dynamic group memberships, nesting group memberships, and static group memberships to determine all the group memberships from a single attribute.

    For example, in IBM Directory Server, you can return all group memberships including the static groups, dynamic groups, and nested groups using the ibm-allGroups attribute. In the Sun ONE directory server you can use the nsRole attribute to calculate, all roles, including managed roles, filtered roles, and nested roles. If an LDAP server has such an attribute in a User object to include dynamic groups, nested groups, and static groups, you can configure WebSphere Application Server security to use this attribute to determine these groups.

    Depending on the implementation, and LDAP server can caluculate dynamic group membership. In this case, this dynamic computation is performed entirely by the LDAP server under a single LDAP query and is invisible to WebSphere Application Server. While this approach is not as efficient as direct groups, server-side dynamic queries are more efficient than determining group membership using static group queries.

    Dynamic group membership, when it is supported by the LDAP server, is frequently reflected back to the LDAP client, which is the WebSphere Application Server. In this configuration, WebSphere Application Server is required to compose the appropriate dynamic query against LDAP for each group. This operation results in 0(zero) groups and does not scale well.

  Tips:

  – Use the efficient direct group membership where possible.

– Use the relatively efficient dynamic group membership where the LDAP computes membership within a single query.
– Use static group membership, or client side dynamic group membership as a secondary alternative. This option only performs well on systems where the number of groups within the LDAP server is "small".

The configurations for the supported, listed LDAP servers are pre-defined to use the optimal group membership mechanisms. They assume that the standard object types and schemas for that LDAP vendor are in use on the LDAP server.

- Evaluate the LDAP registry configuration.
  – **Standalone LDAP registry**

    If you are configuring an LDAP server outside of the list of pre-configured types, you must configure the appropriate value in the Group Member ID map field on the Advanced LDAP Settings panel using the following methods.

    - If you use static group membership, you must specifiy objectclass:attribute pairs. If the objectclass for the group object is, **groupOfUniquePersons**, and within that objectclass, members are listed as **persons**, then the static group membership **Group Member ID map** is **groupOfUniquePersons:persons**.
    - If direct group membership is used, attributes exist in the objectclass, you must use attribute:attribute pairs. For example, if the objectclass for the user is **user**and the objectclasst contains attributes called **ingroup**, which contains each group membership, then the direct group membership **Group Member ID map** is **ingroup:member**.

  – **LDAP Registry within a Federated Repositories Registry**

    If you are configuring an LDAP server outside of the list of pre-configured types, you must configure the appropriate value in the **Group attribute definition** properties for the repository.

    - If static group membership is used, you must specify the name of the object class, and the attribute that is used for indicating membership in **Group attribute definition -> Member attributes**. If the group objectclass for the user is, **groupOfUniquePersons**, and within that objectclass, members are listed as **persons**, then the static group **Member attributes** property is set follows:
      1. In the administrative console, click **Security > Global security**.
      2. Under Available realm definitions, select **Federated repositories**, and then **Configure**. In a multiple security domain environment, click **Security domains > *domain_name***. Under Security Attributes, expand User Realm, and click **Customize for this domain**. Select the Realm type as **Federated repositories** and then click **Configure**.
      3. Under Related items, click **Manage repositories**.
      4. Click **Add** to specify a new external repository or select an external repository that is preconfigured.
      5. Under Additional properties, click **Group attribute definition**.
      6. Under Additional properties, click **Member attributes**.
      7. Click **New** to specify a new member attribute.

      Set the **Name of member attribute** field to `persons`

      Set the **Object class field** to `groupOfUniquePersons`

      When you finish adding or updating your federated repository configuration, go to the **Security > Global security** panel and click **Apply** to validate the changes.
    - If direct group membership is used, then attributes exist in the objectclass for the user and you must use the attribute. For example, if the objectclass for the user is **user**, and it contains attributes called **ingroup** that contain each group membership, then you specify the direct group membership in the **Group attribute definition** property for the repository. Perform the following steps:
      1. In the administrative console, click **Security > Global security**.

2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**.
3. Under Related items, click **Manage repositories**.
4. Click **Add** to specify a new external repository or select an external repository that is preconfigured.
5. Under Additional properties, click **Group attribute definition**.

Set the **Name of group membership attribute** field to `ingroup`.

When you finish adding or updating your federated repository configuration, go to the **Security > Global security** panel and click **Apply** to validate the changes.

- Evaluate Nested Groups.
  - **Nested Groups**

    Depending on the LDAP server implementation, groups can contain only users, or can contain other groups, which is known as a nested group. You configure WebSphere Application Server to properly discover all groups by following this nesting as it applies to either a stand-alone LDAP registry or a LDAP Registry within a Federated Repositories Registry.

    - **Standalone LDAP Registry** The stand-alone LDAP registry default setting performs only a single group membership query. If the groups returned are in fact subgroups of other groups, you must enable the **Perform a nested group search** property on the Advanced LDAP Settings panel of the LDAP registry as follows:
      1. Click **Security > Global security**.
      2. Under User account repository, click the **Available realm definitions** drop-down list, select **Standalone LDAP registry**, and click **Configure**.
      3. Under Additional properties, click **Advanced Lightweight Directory Access Protocol (LDAP) user registry settings**.

      Put a check mark in the **Perform a nested group search** check box.
    - **LDAP Registry within a Federated Repositories Registry** Within Federated repositories, you must configure what you expect the results of the query to return. Based on this information, the Federated repository makes the appropriate calls to establish all group membership. If the LDAP server returns all nested group information within a single direct group query, then you set the **Scope of group membership** attribute property in the group attribute definition to **Nested.** as follows:
      1. In the administrative console, click **Security > Global security**.
      2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**.
      3. Under Related items, click **Manage repositories**.
      4. Click **Add** to specify a new external repository or select an external repository that is preconfigured.
      5. Under Additional properties, click **Group attribute definition**.

      Set the **Scope of group membership** attribute property in the group attribute definition to **Nested.**
    - If the LDAP server returns only the direct membership, then the registry must then make subsequent queries to establish complete membership. To force the Federated Repository to issue subsequent queries, set the **Scope of group membership** attribute property in the **Group attribute definition** for the repository to **Direct**.

Results

While using the direct method, dynamic groups, recursive groups, and static groups can be returned as multiple values of a single attribute. For example, in IBM Directory Server all group memberships, including the static groups, dynamic groups, and nested groups, can be returned using the ibm-allGroups attribute. In Sun ONE, all roles, including managed roles, filtered roles, and nested roles, are calculated

using the nsRole attribute. If an LDAP server can use the nsRole attribute, dynamic groups, nested groups, and static groups are all supported by WebSphere Application Server.

Some LDAP servers do not have recursive computing functionality. For example, although Microsoft Active Directory server has direct group search capability using the memberOf attribute, this attribute lists the groups beneath, which the group is directly nested only and does not contain the recursive list of nested predecessors. The Lotus Domino LDAP server only supports the indirect method to locate the group memberships for a user. You cannot obtain recursive group memberships from a Domino server directly. For LDAP servers without recursive searching capability, WebSphere Application Server security provides a recursive function that is enabled by clicking **Perform a Nested Group Search** in the Advanced LDAP user registry settings. Select this option only if your LDAP server does not provide recursive searches and you want a recursive search.

Configuring dynamic and nested group support for the SunONE or iPlanet Directory Server:

Configure dynamic and nested groups to simplify WebSphere Application Server security management and increase its effectiveness and flexibility.

Before you begin

To use dynamic and nested groups with WebSphere Application Server security, you must be running WebSphere Application Server Version 6.1 or later. Refer to “Dynamic groups and nested group support for LDAP” on page 338 for more information on this topic.

Procedure

1. In the administrative console for WebSphere Application Server, click **Security > Global security**.
2. Under User account repository, click the **Available realm definitions** drop-down list, select **Standalone LDAP registry**, and click **Configure**.
3. Select `SunONE` for the type of LDAP server.
4. Select the **Ignore case for authorization** option.
5. Under Additional Properties, click **Advanced Lightweight Directory Access Protocol (LDAP) user registry settings**.
6. Change the Group filter setting to `&(cn=%v)(objectclass=ldapsubentry))`.
7. Change the Group member ID map setting to `nsRole:nsRole`.
8. Click **Apply** or **OK** to validate the changes.

Configuring dynamic and nested group support for the IBM Tivoli Directory Server:

Configure dynamic and nested groups to simplify WebSphere Application Server security management and increase its effectiveness and flexibility.

Before you begin

When creating groups, ensure that nested and dynamic group memberships work correctly.

Procedure

1. In the administrative console for WebSphere Application Server, click **Security > Global security**.
2. Under User account repository, click **Standalone LDAP registry**, and click **Configure**.
3. Select `IBM Tivoli Directory Server` for the type of LDAP server.
4. Under Additional properties, click **Advanced Lightweight Directory Access Protocol (LDAP) user registry settings**.
5. Change the Group filter value to `(&(cn=%v)(|(objectclass=groupOfNames)(objectclass=groupOfUniqueNames)(objectclass=groupOfURLs)))`.

6. Change the Group member ID map value to `ibm-allGroups:member;ibm-allGroups:uniqueMember`.
7. Click **Apply** or **OK** to validate the changes.
8. Verify that Auxiliary object class field on the Add an LDAP entry panel for your IBM Tivoli Directory server has the appropriate value. When you create a nested group, the Auxiliary object class value is `ibm-nestedGroup`. When you create a dynamic group, the Auxiliary object class value is `ibm-dynamicGroup`.

Configuring multiple LDAP servers for user registry failover

WebSphere Application Server security can be configured to attempt failovers between multiple Lightweight Directory Access Protocol (LDAP) hosts.

Before you begin

The multiple LDAP servers involved in the failover can be replicas that are replicated from the same master LDAP server, or they can be any LDAP host with the same schema. That is any LDAP host that contains data that is imported from the same LDAP data interchange format (LDIF) file.

Note: When WebSphere Application Server attempts failovers between multiple Lightweight Directory Access Protocol (LDAP) hosts, system properties are exchanged. WebSphere Application Server Version 6.1.0 manages the SSL configuration and these system properties. You cannot expect to set system properties yourself and expect the failover to succeed.

Procedure

1. Start the application server process.
   a. Start the Command Prompt application.
   b. Change directories to *profile_root*\bin.
   c. Enter **startServer**.
2. Start the wsadmin Command Prompt application.
   a. Start the Command Prompt application.
   b. Change directories to *profile_root*\bin.
   c. Enter the following command:
      ```
      wsadmin –user username –password password
      ```
3. Configure a second LDAP server for failover.
   a. Enter the following command to set the failover LDAP server hostname:
      ```
      set ldapServer [ldap server hostname]
      ```
   b. Enter the following command to set the LDAP server port number:
      ```
      set ldapPort [ldap server port]
      ```
   c. Enter the following command to set the WebSphere LDAP failover variable:
      ```
      set Attrs2 [list [list hosts [list [list [list host $ldapServer] [list port $ldapPort]]]]]
      ```
   d. Modify the LDAP configuration to add the failover LDAP server by entering the following command:
      ```
      set result [$AdminConfig list LDAPUserRegistry]
      ```
   e. Find the LDAP server configID by entering the following command:
      ```
      $AdminConfig modify $result $Attrs2
      ```
   f. Enter the following command to save the configuration change:
      ```
      $AdminConfig save
      ```
   g. Enter **exit** to quit the Command Prompt application. The following is an example of the Command Prompt application output:
      ```
      wsadmin>set ldapServer [list xxxx.xxxx.xxx.com]
      xxxx.xxxx.xxx.com
      wsadmin>set ldapPort [list NNN]
      NNN
      ```

```
wsadmin>set Attrs2 [list [list hosts [list [list [list host $ldapServer] [list port $ldapPort]]]]]
{hosts {{{host xxxx.xxxx.xxx.com} {port NNN}}}}
wsadmin> set result [$AdminConfig list LDAPUserRegistry]
(cells/Father2Cell01|security.xml#LDAPUserRegistry_1)
wsadmin>$AdminConfig modify $result $Attrs2

wsadmin>$AdminConfig save
```

4. Review the configuration change by opening the `security.xml` file with a text editor and review the new entry.
5. Stop the application server.
   a. Start the Command Prompt application.
   b. Change directories to *profile_root*\bin.
   c. To stop the application server, enter the following command:

      ```
      stopServer –user username –password password
      ```

Testing an LDAP server for user registry failover

After configuring a Lightweight Directory Access Protocol (LDAP) host for failover you should test the failover server by stopping the main LDAP server.

Before you begin

This task assumes the following setup:

- Deployment Manager is installed on the primary LDAP server running Application Server version 6.0.2 or higher.
- All other LDAP hosts are Active Directory machines with similar user registry designs.
- Atleast one of the other LDAP hosts has been configured for failover.

Note: This topic references one or more of the application server log files. As a recommended alternative, you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log` , `SystemErr.log`, `trace.log`, and `activity.log` files on distributed and IBM i systems. You can also use HPEL in conjunction with your native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

Procedure

1. Stop the Active Directory Server on the failover server.
2. Start the deployment manager process.
   a. Start the Command Prompt application.
   b. Change directories to *profile_root*\bin.
   c. Enter **startManager**.
3. Review the `SystemOut.log` file to see if the LDAP failover happened. The sample text is an example of a `SystemOut.log` file that records a successful failover:

   ```
   [7/11/05 15:38:31:324 EDT] 0000000a LdapRegistryI A   SECJ0418I:
   Cannot connect to the LDAP server ldap://xxxx.xxxxx.xxxx.com:NNN. {primary LDAP server}
   [7/11/05 15:38:32:486 EDT] 0000000a UserRegistryI A   SECJ0136I:
   Custom Registry:com.ibm.ws.security.registry.ldap.LdapRegistryImpl has been initialized
   [7/11/05 15:38:53:787 EDT] 0000000a LdapRegistryI A   SECJ0419I:
   The user registry is currently connected to the LDAP server ldap://xxxx.xxxxx.xxxx.com:NNN. {failover LDAP server}
   ...
   [7/11/05 15:39:35:667 EDT] 0000000a WsServerImpl  A   WSVR0001I: Server dmgr open for e-business
   ```

4. Log into the console to see working and non-working cases.
   a. Start a browser.
   b. Browse to `http://localhost:9060/admin`.

c. Type in your user ID and password and click **OK**.
d. Log out of the Administrative Console.
e. Type in `DummyAdmin` as the user ID and `dummy1admin` as your password and click **OK**. This should fail proving WebSphere Application Server is connected to the other LDAP server. Please make sure that on a production system the user registries are identical so this problem does not happen when switching between LDAP servers.

5. Stop the deployment manager.
   a. Start the Command Prompt application.
   b. Change directories to *profile_root*\bin.
   c. To stop the deployment manager, enter the following command:

```
stopManager –user username –password password
```

Deleting LDAP endpoints using wsadmin

You can delete Lightweight Directory Access Protocol (LDAP) endpoints for a user registry by using the WebSphere Application Server administrative tool (wsadmin).

Procedure

1. Start the wsadmin scripting tool.
2. Set the LDAP variable and display a list of LDAP endpoint objects. Enter the following commands:
   Using Jacl:

```
set ldap [$AdminConfig list LDAPUserRegistry]

$AdminConfig list EndPoint $ldap
```

   Using Jython:

```
ldap=AdminConfig.list["LDAPUserRegistry"]

print AdminConfig.show(ldap)
```

   For the Jython language, you can obtain the endpoint from the host variable after running the previous command.
3. Display a list of LDAP endpoint objects. Enter the following command for each object:
   Using Jacl:

```
$AdminConfig showall End_Point_Object
```

   Using Jython:

```
AdminConfig.showall("End_Point_Object")
```

4. Delete an LDAP endpoint object. Enter the following command:
   Using Jacl:

```
$AdminConfig remove End_Point_Object
```

   Using Jython:

```
AdminConfig.remove ("End_Point_Object")
```

5. Save your configuration changes: Enter the following command:
   Using Jacl:

```
$AdminConfig save
```

   Using Jython:

```
AdminConfig.save()
```

Updating LDAP binding information

Use this information to dynamically update security LDAP binding information by switching to a different binding identity.

About this task

You can dynamically update Lightweight Directory Access Protocol (LDAP) binding information without first stopping and restarting WebSphere Application Server by using the **wsadmin** tool.

The resetLdapBindInfo method in SecurityAdmin MBean is used to dynamically update LDAP binding information at WebSphere Application Server security runtime, and it takes the bind distinguished name (DN) and bind password parameters as input. The resetLdapBindInfo method validates the bind information against the LDAP server. If validation passes, new binding information is stored in `security.xml`, and a copy of the information is placed in WebSphere Application Server security runtime.

If the new binding information is `null`, `null`, the resetLdapBindInfo method first extracts LDAP binding information, including bind DN, bind password, and target binding host from WebSphere Application Server security configuration in `security.xml`. It then pushes the binding information to WebSphere Application Server security runtime.

There are two ways to dynamically update WebSphere Application Server security LDAP binding information using the SecurityAdmin MBean through wsadmin:

- "Switching to a different binding identity"
- "Switching to a failover LDAP host"

Switching to a different binding identity:
About this task

To dynamically update security LDAP binding information by switching to a different binding identity:

Procedure

1. In the administrative console, click **Security > Global security**.
2. Under User account repository, click the **Available realm definitions** drop-down list, select **Standalone LDAP registry**, and click **Configure**.
3. Create a new bind DN. It must have the same access authority as the current bind DN.
4. Run the SecurityAdmin MBean across all of the application server processes to validate the new binding information, to save it to `security.xml`, and to push the new binding information to the runtime.

Example

The following is a sample Jacl file for step 4:

```
proc LDAPReBind {args} {
  global AdminConfig AdminControl ldapBindDn ldapBindPassword
  set ldapBindDn [lindex $args 0]
  set ldapBindPassword [lindex $args 1]
       set secMBeans [$AdminControl queryNames type=SecurityAdmin,*]
       set plist  [list $ldapBindDn $ldapBindPassword]
       foreach secMBean $secMBeans {
            set result [$AdminControl invoke $secMBean resetLdapBindInfo $plist]
       }
 }
```

Switching to a failover LDAP host:
About this task

To dynamically update security LDAP binding information by switching to a failover LDAP host:

Procedure

1. In the administrative console, click **Security > Global security**.
2. Under User account repository, select **Standalone LDAP registry** and click **Configure**.
3. Change the password for bind DN on one LDAP server (it can be the primary or the backup).

4. Update the new bind DN password to WebSphere Application security runtime by calling resetLdapBindInfo with the bind DN and by using its new password as a parameter.
5. Use the new bind DN password for all of the other LDAP servers. The binding information is now consistent across WebSphere Application Server and the LDAP servers.

   If you call resetLdapBindInfo with `null`, `null` as input parameters, WebSphere Application Server security runtime completes the following steps:

   a. Reads the bind DN, bind password, and target LDAP hosts from `security.xml`.

   b. Refreshes the cached connection to the LDAP server.

   If you configure security to use multiple LDAP servers, this MBean call forces WebSphere Application Server security to reconnect to the first available LDAP host in the list. For example, if three LDAP servers are configured in the order of L1, L2, and L3, the reconnection process always starts with the L1 server.

   When LDAP failover is configured by associating a single hostname to multiple IP addresses, entering an invalid password can cause multiple LDAP bind retries. With the default settings, the number of LDAP bind retries is equal to one more than the number of associated IP addresses. This means a single invalid login attempt can cause the LDAP account to be locked. If the com.ibm.websphere.security.registry.ldap.singleLDAP custom property is set to false, LDAP bind calls are not retried.

   When LDAP failover is configured by registering backend LDAP server hostnames using wsadmin command, set the com.ibm.websphere.security.ldap.retryBind property to false.

   gotcha: Federated repository does not support failover by associating a single hostname to multiple IP addresses. This feature is only available in stand-alone LDAP.

Configuring stand-alone custom registries

Use the following information to configure stand-alone custom registries through the administrative console.

Before you begin

Before you begin this task, implement and build the UserRegistry interface. For more information on developing stand-alone custom registries refer to “Developing stand-alone custom registries” on page 823. The following steps are required to configure stand-alone custom registries through the administrative console.

Procedure

1. Click **Security > Global security**.
2. Under User account repositories, select **Stand-alone custom registry** and click **Configure**.
3. Enter a valid user name in the Primary administrative user name field. This ID is the security server ID, which is only used for WebSphere Application Server security and is not associated with the system process that runs the server. The server calls the local operating system registry to authenticate and obtain privilege information about users by calling the native APIs in that particular registry.
4. Enter the dot-separated class name that implements the com.ibm.websphere.security.UserRegistry interface in the Custom registry class name field. For the sample, this file name is `com.ibm.websphere.security.FileRegistrySample`.

   Attention: The sample provided is intended to familiarize you with this feature. Do not use this sample in an actual production environment.
5. Add your custom registry class name to the class path. It is recommended that you add the Java Archive (JAR) file that contains your custom user registry implementation to the following directory:
   - *app_server_root*`/lib/ext`

6. Optional: Select the **Ignore case for authorization** option for the authorization to perform a case insensitive check. Enabling this option is necessary only when your user registry is case insensitive and does not provide a consistent case when queried for users and groups.
7. Click **Apply** if you have any other additional properties to enter for the registry initialization.
8. Optional: Enter additional properties to initialize your implementation.
   a. Click **Custom properties > New**.
   b. Enter the property name and value.

      For the sample, enter the following two properties. It is assumed that the `users.props` file and the `groups.props` file are in the *customer_sample* directory under the product installation directory. You can place these properties in any directory that you choose and reference their locations through custom properties. However, make sure that the directory has the appropriate access permissions.

Table 16. Additional properties.

This table lists additional custom properties when configuring stand-alone custom registries.

Property name	Property value
usersFile	${*USER_INSTALL_ROOT*}/*customer_sample* /users.props
groupsFile	${*USER_INSTALL_ROOT*}/*customer_sample* /groups.props

      Samples of these two properties are available in “users.props file” on page 219 and “groups.props file” on page 220.

      The **Description**, **Required**, and **Validation Expression** fields are not used and can remain blank.

      WebSphere Application Server version 4-based custom user registry is migrated to the custom user registry based on the com.ibm.websphere.security.UserRegistry interface.
   c. Click **Apply**.
   d. Repeat this step to add other additional properties.
9. Click **Security > Global security**.
10. Under User account repository, click the **Available realm definitions** drop-down list, select **Stand-alone custom registry**, and click **Configure**.
11. Select either the **Automatically generated server identity** or **Server identity that is stored in the repository** option. If you select the **Server identity that is stored in the repository** option, enter the following information:

    Server user ID or administrative user on a Version 6.0.x node
    Specify the short name of the account that is chosen in the second step.

    Server user password
    Specify the password of the account that is chosen in the second step.
12. Click **OK** and complete the required steps to turn on security.

Results

This set of steps is required to set up the stand-alone custom registry and to enable security in WebSphere Application Server.

Note: The security component of WebSphere Application Server expands a selected list of variables when enabling security. See the information about variable settings for more details.

What to do next

1. Complete the remaining steps, if you are enabling security.
2. Validate the user and password. Save and synchronize in the cell environment.

3. After security is turned on, save, stop, and start all the product servers, including cell, nodes, and all of the application servers, for any changes to take effect. If the server comes up without any problems, the setup is correct.

Stand-alone custom registries

A *stand-alone custom registry* is a customer-implemented registry that implements the UserRegistry Java interface, as provided by the product. A custom-implemented registry can support virtually any type of an account repository from a relational database, flat file, and so on. The custom user registry provides considerable flexibility in adapting product security to various environments where some form of a registry or repository other than federated repositories, stand-alone Lightweight Directory Access Protocol (LDAP) registry or local operating system registry already exists in the operational environment.

WebSphere Application Server security provides an implementation that uses various local operating system-based registries and various stand-alone Lightweight Directory Access Protocol (LDAP)-based registries. However, situations can exist where your user and group data resides in other repositories or custom user registries, such as a database, and moving this information to either a local operating system registry or a stand-alone LDAP registry implementation might not be feasible. For these situations, WebSphere Application Server security provides a service provider interface (SPI) that you can implement to interact with your current registry. The custom registry feature supports any user registry that is not implemented by WebSphere Application Server.

The SPI is the UserRegistry interface. The *UserRegistry interface* is a collection of methods that are required for authorization purposes. These methods authenticate individual users using either a password or certificates and collect information about the user, which are called privilege attributes. This interface also includes methods that obtain user and group information so that they can be given access to resources. When implementing the methods in the interface, you must decide how to map the information that is manipulated by the UserRegistry interface to the information in your registry.

This interface has a set of methods to implement for the product security to interact with your registries for all security-related tasks. The local operating system and LDAP registry implementations that are provided also implement this interface. Custom user registries are sometimes called the *pluggable user registries* or *custom registries* for short. Your custom user registry implementation is expected to be thread-safe.

Building a custom registry is a software implementation effort. The implementation does not depend on other WebSphere Application Server resources, for example, data sources, for its operation.

Make sure that your implementation of the custom registry does not depend on any WebSphere Application Server components such as data sources, enterprise beans, and so on. Do not have this dependency because security is initialized and enabled prior to most of the other WebSphere Application Server components during startup. If your previous implementation used these components, make a change that eliminates the dependency.

The methods in the UserRegistry interface operate on the following information for users:

User security name

The user name is similar to the user name in the local operating system registries.

This name is used to log in when prompted by a secured application. By default, the Enterprise JavaBeans (EJB) getCallerPrincipal method and the getRemoteUser and getUserPrincipal servlet methods return this name. The user security name is also referred to as *userSecurityName*, *userName*, or *user name*.

WAS_UseDisplayName

This is a custom property of User Registries. This property defines the returning value of the getCallerPrincipal(), getUserPrincipal(), and RemoteUser() methods. The following shows acceptable values for WAS_UseDisplayName:

- `false` This is default. Security Name is returned.
- `true` The display name is returned. This setting requires that the custom property **com.ibm.websphere.security.useLoggedSecurityName** be set to true.

Unique user ID
This ID represents a unique identifier for the user, which is required by the UserRegistry interface. The unique ID is similar to the system ID (SID) in Windows systems, the Unique ID (UID) in Linux and UNIX systems, and the distinguished name (DN) in Lightweight Directory Authentication Protocol (LDAP). This ID is also referred to as *uniqueUserId*. The unique ID is used to make the authorization decisions for protected resources.

Display user name
This name is an optional string that describes a user. The implementation can use display names for informational purposes only; these names are not required to exist or to be unique. The user interface can use the display name to present more information about the user.

Group security name
This name, which represents the security group, is also referred to as *groupSecurityName*, *groupName*, and *group name*.

Unique group ID
The unique ID is the identifier for a group. This name is also referred to as *uniqueGroupId* ID.

Display group name
The display name is an optional string that describes a group.

The topic on UserRegistry interface describes each of the methods in the interface that need implementing. An explanation of each of the methods and their usage in the sample and any changes from the Version 4 interface are provided. The Related references section provides links to all other custom user registries documentation, including a file-based registry sample. The Sample provided is very simple and is intended to familiarize you with this feature. Do not use this sample in an actual production environment.

Stand-alone custom registry settings

Use this page to configure the stand-alone custom registry.

To view this administrative console page, complete the following steps:

1. Click **Security > Global security**.
2. Under User account repository, click the **Available realm definitions** drop-down list, select **Stand-alone custom registry**, and click **Configure**.

After the properties are set in this panel, click **Apply**. Under Additional Properties, click **Custom properties** to include additional properties that the custom user registry requires.

Note: Custom properties might include information such as specifying lists of users or groups.

When security is enabled and any of these custom user registry settings change, go to the Global security panel and click **Apply** to validate the changes.

WebSphere Application Server Version 7.0 distinguishes between the user identities for administrators who manage the environment and server identities for authenticating server to server communications. In most cases, server identities are automatically generated and are not stored in a repository.

However, if you are adding a previous version node to the latest version cell and the previous version node used a server identity and password, you must ensure that the server identity and password for the previous version are defined in the repository for this cell. Enter the server user identity and password on this panel.

Primary administrative user name:

Specifies the name of a user with administrative privileges that is defined in your custom user registry.

The user name is used to log onto the administrative console when administrative security is enabled. Version 6.1 requires an administrative user that is distinct from the server user identity so that administrative actions can be audited.

Attention: In WebSphere Application Server, Version 6.0.x, a single user identity is required for both administrative access and internal process communication. When migrating to Version 6.1 and above, this identity is used as the server user identity. You need to specify another user for the administrative user identity.

Automatically generated server identity:

Enables the application server to generate the server identity, which is recommended for environments that contain only Version 6.1 or later nodes. Automatically generated server identities are not stored in a user repository.

Information	Value
Default:	Enabled

Server identity that is stored in the repository:

Specifies a user identity in the repository that is used for internal process communication. Cells that contain Version 6.1 or later nodes require a server user identity that is defined in the active user repository.

Information	Value
Default:	Enabled

Server user ID or administrative user on a Version 6.0.x node:

Specifies the user ID that is used to run the application server for security purposes.

Password:

Specifies the password that corresponds to the server ID.

Custom registry class name:

Specifies a dot-separated class name that implements the com.ibm.websphere.security.UserRegistry interface.

Put the custom registry class name in the class path. A suggested location is the following directory.

- *%install_root%*/lib/ext

Information	Value
Data type:	String
Default:	com.ibm.websphere.security.FileRegistrySample

Ignore case for authorization:

Indicates that a case-insensitive authorization check is performed when you use the default authorization.

Information	Value
Default:	Disabled
Range:	Enabled or Disabled

Stand-alone custom registry wizard settings

A wizard page exists in the administrative console to aid in viewing the basic settings necessary to connect the application server to an existing stand-alone custom registry. After you have viewed the basic settings, you can also modify the existing stand-alone customer registry configuration using the administrative console.

To view this security wizard page, complete the following steps:

1. Click **Security > Global security > Security configuration wizard**.
2. Select your protection settings and click **Next**.
3. Select the **Stand-alone custom registry** option and click **Next**.

You can modify your stand-alone custom registry configuration by completing the following steps:

1. Click **Security > Global security**.
2. Under User account repository, click the **Available realm definitions** drop-down list, select **Stand-alone custom registry**, and click **Configure**.
3. Enter additional properties to initialize your implementation
   - Click **Custom properties > New**.
   - Enter the property name and value. For the sample, enter the following two properties. It is assumed that the `users.props` file and the `groups.props` file are in the *customer_sample* directory under the product installation directory. You can place these properties in any directory that you choose and reference their locations through Custom properties. However, make sure that the directory has the appropriate access permissions.

Table 17. Custom properties.

This table lists additional custom properties when changing stand-alone custom registry wizard settings.

Property name	Property value
usersFile	${*USER_INSTALL_ROOT*}/*customer_sample* /users.props
groupsFile	${*USER_INSTALL_ROOT*}/*customer_sample* /groups.props

Samples of these two properties are available in reference topics for the `users.props` file and the `groups.props` file. See the related links below for more information.

The Description, Required, and Validation Expression fields are not used and can remain blank.

WebSphere Application Server Version 4 based custom user registry is migrated to the custom user registry based on the com.ibm.websphere.security.UserRegistry interface.

- Click **Apply**.

Primary administrative user name:

Specifies the name of a user with administrative privileges that is defined in your custom user registry.

The user name is used to log onto the administrative console when administrative security is enabled. Version 6.1 requires an administrative user that is distinct from the server user identity so that administrative actions can be audited.

Attention: In WebSphere Application Server, Version 6.0.x, a single user identity is required for both administrative access and internal process communication. When migrating to Version 6.1 and above, this identity is used as the server user identity. You need to specify another user for the administrative user identity.

Custom registry class name:

Specifies a dot-separated class name that implements the com.ibm.websphere.security.UserRegistry interface.

Put the custom registry class name in the class path. A suggested location is the following directory.

- *%install_root%*/lib/ext

Information	Value
Data type:	String
Default:	com.ibm.websphere.security.FileRegistrySample

Ignore case for authorization:

Indicates that a case-insensitive authorization check is performed when you use the default authorization.

Information	Value
Default:	Disabled
Range:	Enabled or Disabled

FileRegistrySample.java file

This provides an example of the `FileRegistrySample.java` file.

The user and group information required by this sample is contained in the “users.props file” on page 219 and “groups.props file” on page 220 files.

Attention: The samples that are provided are intended to familiarize you with this feature. Do not use these samples in an actual production environment.

The contents of the `FileRegistrySample.java` file:

```
//
// 5639-D57, 5630-A36, 5630-A37, 5724-D18
// (C) COPYRIGHT International Business Machines Corp. 1997, 2005
// All Rights Reserved * Licensed Materials - Property of IBM
//

//-----------------------------------------------------------------------
// This program may be used, run, copied, modified and distributed
// without royalty for the purpose of developing, using, marketing, or
// distributing.
//-----------------------------------------------------------------------
//

// This sample is for the custom user registry feature in WebSphere Application Server.

import java.util.*;
import java.io.*;
import java.security.cert.X509Certificate;
import com.ibm.websphere.security.*;

/**
 *  The main purpose of this sample is to demonstrate the use of the
 *  custom user registry feature available in WebSphere Application Server. This
 *  sample is a file-based registry sample where the users and the groups
 *  information is listed in files (users.props and groups.props). As such
 *  simplicity and not the performance was a major factor. This
 *  sample should be used only to get familiarized with this feature. An
 *  actual implementation of a realistic registry should consider various
 *  factors like performance, scalability, thread safety, and so on.
 **/
public class FileRegistrySample implements UserRegistry {

   private static String USERFILENAME = null;
```

```
 private static String GROUPFILENAME = null;

 /** Default Constructor **/
 public FileRegistrySample() throws java.rmi.RemoteException {
 }

/**
 * Initializes the registry. This method is called when creating the
 * registry.
 *
 * @param     props - The registry-specific properties with which to
 *                    initialize the custom registry
 * @exception CustomRegistryException
 *                    if there is any registry-specific problem
 **/
public void initialize(java.util.Properties props)
       throws CustomRegistryException {
    try {

       /* try getting the USERFILENAME and the GROUPFILENAME from
        * properties that are passed in (For example, from the
        * administrative console). Set these values in the administrative
        * console. Go to the special custom settings in the custom
        * user registry section of the Authentication panel.
        * For example:
        * usersFile   c:/temp/users.props
        * groupsFile  c:/temp/groups.props
        */
        if (props != null) {
           USERFILENAME = props.getProperty("usersFile");
           GROUPFILENAME = props.getProperty("groupsFile");
        }

    } catch(Exception ex) {
       throw new CustomRegistryException(ex.getMessage(),ex);
    }

    if (USERFILENAME == null || GROUPFILENAME == null) {
       throw new CustomRegistryException("users/groups information missing");
    }

 }

/**
 * Checks the password of the user. This method is called to authenticate
 * a user when the user's name and password are given.
 *
 * @param  userSecurityName the name of user
 * @param  password the password of the user
 * @return a valid userSecurityName. Normally this is
 *         the name of same user whose password was checked
 *         but if the implementation wants to return any other
 *         valid userSecurityName in the registry it can do so
 * @exception CheckPasswordFailedException if userSecurityName/
 *            password combination does not exist
 *            in the registry
 * @exception CustomRegistryException if there is any registry-
 *            specific problem
 **/
 public String checkPassword(String userSecurityName, String passwd)
    throws PasswordCheckFailedException,
           CustomRegistryException {
    String s,userName = null;
```

```
    BufferedReader in = null;

    try {
       in = fileOpen(USERFILENAME);
       while ((s=in.readLine())!=null)
       {
          if (!(s.startsWith("#") || s.trim().length() <=0 )) {
             int index = s.indexOf(":");
             int index1 = s.indexOf(":",index+1);
             // check if the userSecurityName:passwd combination exists
             if ((s.substring(0,index)).equals(userSecurityName) &&
                    s.substring(index+1,index1).equals(passwd)) {
                // Authentication successful, return the userID.
                userName = userSecurityName;
                break;
             }
          }
       }
    } catch(Exception ex) {
       throw new CustomRegistryException(ex.getMessage(),ex);
    } finally {
       fileClose(in);
    }


    if (userName == null) {
       throw new PasswordCheckFailedException("Password check failed for user:"
       + userSecurityName);
    }

    return userName;
 }

/**
 * Maps an X.509 format certificate to a valid user in the registry.
 * This is used to map the name in the certificate supplied by a browser
 * to a valid userSecurityName in the registry
 *
 * @param     cert the X509 certificate chain
 * @return    The mapped name of the user userSecurityName
 * @exception CertificateMapNotSupportedException if the
 *            particular certificate is not supported.
 * @exception CertificateMapFailedException if the mapping of
 *            the certificate fails.
 * @exception CustomRegistryException if there is any registry
 *            -specific problem
 **/
 public String mapCertificate(X509Certificate[] cert)
    throws CertificateMapNotSupportedException,
           CertificateMapFailedException,
           CustomRegistryException {
    String name=null;
    X509Certificate cert1 = cert[0];
    try {
       // map the SubjectDN in the certificate to a userID.
       name = cert1.getSubjectDN().getName();
    } catch(Exception ex) {
       throw new CertificateMapNotSupportedException(ex.getMessage(),ex);
    }

    if(!isValidUser(name)) {
       throw new CertificateMapFailedException("user:"  + name
       +  "is not valid");
```

```
    }
    return name;
 }

/**
 * Returns the realm of the registry.
 *
 * @return the realm. The realm is a registry-specific string
 * indicating the realm or domain for which this registry
 * applies. For example, for OS/400 or AIX this would be
 * the host name of the system whose user registry this
 * object represents. If null is returned by this method,
 * realm defaults to the value of "customRealm". It is
 * recommended that you use your own value for realm.
 *
 * @exception CustomRegistryException if there is any registry-
 * specific problem
 **/
 public String getRealm()
    throws CustomRegistryException {
    String name = "customRealm";
    return name;
 }

/**
 * Gets a list of users that match a pattern in the registry.
 * The maximum number of users returned is defined by the limit
 * argument.
 * This method is called by the administrative console and scripting
 * (command line) to make the users in the registry available for
 * adding them (users) to roles.
 *
 * @param      pattern the pattern to match. (For example, a* will
 *             match  all userSecurityNames starting with a)
 * @param      limit the maximum number of users that should be
 *             returned. This is very useful in situations where
 *             there are thousands of users in the registry and
 *             getting all of them at once is not practical. The
 *             default is 100. A value of 0 implies get all the
 *             users and hence must be used with care.
 * @return     a Result object that contains the list of users
 *             requested and a flag to indicate if  more users
 *             exist.
 * @exception  CustomRegistryException if there is any registry-
 *             specific problem
 **/
 public Result getUsers(String pattern, int limit)
    throws CustomRegistryException {
    String s;
    BufferedReader in = null;
    List allUsers = new ArrayList();
    Result result = new Result();
    int count = 0;
    int newLimit = limit+1;
    try {
       in = fileOpen(USERFILENAME);
       while ((s=in.readLine())!=null)
       {
          if (!(s.startsWith("#") || s.trim().length() <=0 )) {
             int index = s.indexOf(":");
             String user = s.substring(0,index);
             if (match(user,pattern)) {
                allUsers.add(user);
```

```
                if (limit !=0 && ++count == newLimit) {
                   allUsers.remove(user);
                   result.setHasMore();
                   break;
                }
             }
          }
       }
    } catch (Exception ex) {
       throw new CustomRegistryException(ex.getMessage(),ex);
    } finally {
       fileClose(in);
    }

    result.setList(allUsers);
    return result;
 }

/**
 * Returns the display name for the user specified by
 *  userSecurityName.
 *
 * This method may be called only when the user information
 * is displayed (information purposes only, for example, in
 * the administrative console) and hence not used in the actual
 * authentication or authorization purposes. If there are no
 * display names in the registry return null or empty string.
 *
 * In WebSphere Application Server 4.x custom registry, if you
 * had a display name for the user and if it was different from the
 * security name, the display name was returned for the EJB
 * methods getCallerPrincipal() and the servlet methods
 * getUserPrincipal() and  getRemoteUser().
 * In  Version 5.x and later, for the
 * same methods, the security name will be returned by default.
 * This is the recommended way as the display name is not unique
 * and might create security holes. However, for backward
 * compatibility if you need the display name to be returned
 * set the property WAS_UseDisplayName to true.
 *
 *See the Information Center documentation for more information.
 *
 * @param     userSecurityName the name of the user.
 * @return    the display name for the user. The display
 *            name is a registry-specific string that
 *            represents a descriptive, not necessarily
 *            unique, name for a user. If a display name
 *            does not exist return null or empty string.
 * @exception EntryNotFoundException if userSecurityName
 *            does not exist.
 * @exception CustomRegistryException if there is any registry-
 *  *         specific problem
 **/
 public String getUserDisplayName(String userSecurityName)
    throws CustomRegistryException,
           EntryNotFoundException {

    String s,displayName = null;
    BufferedReader in = null;

    if(!isValidUser(userSecurityName)) {
       EntryNotFoundException nsee = new EntryNotFoundException("user:"
       + userSecurityName +  "is not valid");
```

```
            throw nsee;
        }

        try {
            in = fileOpen(USERFILENAME);
            while ((s=in.readLine())!=null)
            {
                if (!(s.startsWith("#") || s.trim().length() <=0 )) {
                    int index = s.indexOf(":");
                    int index1 = s.lastIndexOf(":");
                    if ((s.substring(0,index)).equals(userSecurityName)) {
                        displayName = s.substring(index1+1);
                        break;
                    }
                }
            }
        } catch(Exception ex) {
            throw new CustomRegistryException(ex.getMessage(), ex);
        } finally {
            fileClose(in);
        }

        return displayName;
    }

    /**
     * Returns the unique ID for a userSecurityName. This method is called
     * when creating a credential for a user.
     *
     * @param    userSecurityName - The name of the user.
     * @return   The unique ID of the user. The unique ID for a user
     *           is the stringified form of some unique, registry-specific,
     *           data that serves to represent the user. For example, for
     *           the UNIX user registry, the unique ID for a user can be
     *           the UID.
     * @exception EntryNotFoundException if userSecurityName does not
     *            exist.
     * @exception CustomRegistryException if there is any registry-
     *            specific problem
     **/
     public String getUniqueUserId(String userSecurityName)
        throws CustomRegistryException,
               EntryNotFoundException {

        String s,uniqueUsrId = null;
        BufferedReader in = null;
        try {
            in = fileOpen(USERFILENAME);
            while ((s=in.readLine())!=null)
            {
                if (!(s.startsWith("#") || s.trim().length() <=0 )) {
                    int index = s.indexOf(":");
                    int index1 = s.indexOf(":", index+1);
                    if ((s.substring(0,index)).equals(userSecurityName)) {
                        int index2 = s.indexOf(":", index1+1);
                        uniqueUsrId = s.substring(index1+1,index2);
                        break;
                    }
                }
            }
        } catch(Exception ex) {
            throw new CustomRegistryException(ex.getMessage(),ex);
        } finally {
```

```
            fileClose(in);
        }

        if (uniqueUsrId == null) {
            EntryNotFoundException nsee =
            new EntryNotFoundException("Cannot obtain uniqueId for user:"
            + userSecurityName);
            throw nsee;
        }

        return uniqueUsrId;
    }

    /**
     * Returns the name for a user given its unique ID.
     *
     * @param      uniqueUserId  - The unique ID of the user.
     * @return     The userSecurityName of the user.
     * @exception  EntryNotFoundException if the unique user ID does not exist.
     * @exception  CustomRegistryException if there is any registry-specific
     *             problem
     **/
     public String getUserSecurityName(String uniqueUserId)
        throws CustomRegistryException,
               EntryNotFoundException {
        String s,usrSecName = null;
        BufferedReader in = null;
        try {
            in = fileOpen(USERFILENAME);
            while ((s=in.readLine())!=null)
            {
               if (!(s.startsWith("#") || s.trim().length() <=0 )) {
                  int index = s.indexOf(":");
                  int index1 = s.indexOf(":", index+1);
                  int index2 = s.indexOf(":", index1+1);
                  if ((s.substring(index1+1,index2)).equals(uniqueUserId)) {
                     usrSecName = s.substring(0,index);
                     break;
                  }
               }
            }
        } catch (Exception ex) {
            throw new CustomRegistryException(ex.getMessage(), ex);
        } finally {
            fileClose(in);
        }

        if (usrSecName == null) {
            EntryNotFoundException ex =
               new EntryNotFoundException("Cannot obtain the
               user securityName for"  + uniqueUserId);
            throw ex;
        }

        return usrSecName;

     }

    /**
     * Determines if the userSecurityName exists in the registry
     *
     * @param     userSecurityName - The name of the user
     * @return    True if the user is valid; otherwise false
```

```
 * @exception CustomRegistryException if there is any registry-
 *            specific problem
 * @exception RemoteException as this extends java.rmi.Remote
 *            interface
 **/
 public boolean isValidUser(String userSecurityName)
    throws CustomRegistryException {
    String s;
    boolean isValid = false;
    BufferedReader in = null;
    try {
       in = fileOpen(USERFILENAME);
       while ((s=in.readLine())!=null)
       {
          if (!(s.startsWith("#") || s.trim().length() <=0 )) {
             int index = s.indexOf(":");
             if ((s.substring(0,index)).equals(userSecurityName)) {
                isValid=true;
                break;
             }
          }
       }
    } catch (Exception ex) {
       throw new CustomRegistryException(ex.getMessage(), ex);
    } finally {
       fileClose(in);
    }

    return isValid;
 }


/**
 * Gets a list of groups that match a pattern in the registry
 * The maximum number of groups returned is defined by the
 * limit argument. This method is called by administrative console
 * and scripting (command line) to make available the groups in
 * the registry for adding them (groups) to roles.
 *
 * @param       pattern the pattern to match. (For example, a* matches
 *              all groupSecurityNames starting with a)
 * @param       Limits the maximum number of groups to return
 *              This is very useful in situations where there
 *              are thousands of groups in the registry and getting all
 *              of them at once is not practical. The default is 100.
 *              A value of 0 implies get all the groups and hence must
 *              be used with care.
 * @return      A Result object that contains the list of groups
 *              requested and a flag to indicate if more groups exist.
 * @exception CustomRegistryException if there is any registry-specific
 *              problem
 **/
 public Result getGroups(String pattern, int limit)
    throws CustomRegistryException {
    String s;
    BufferedReader in = null;
    List allGroups = new ArrayList();
    Result result = new Result();
    int count = 0;
    int newLimit = limit+1;
    try {
       in = fileOpen(GROUPFILENAME);
       while ((s=in.readLine())!=null)
```

```
      {
         if (!(s.startsWith("#") || s.trim().length() <=0 )) {
            int index = s.indexOf(":");
            String group = s.substring(0,index);
            if (match(group,pattern)) {
               allGroups.add(group);
               if (limit !=0 && ++count == newLimit) {
                  allGroups.remove(group);
                  result.setHasMore();
                  break;
               }
            }
         }
      }
   } catch (Exception ex) {
      throw new CustomRegistryException(ex.getMessage(),ex);
   } finally {
      fileClose(in);
   }

   result.setList(allGroups);
   return result;
}

/**
 * Returns the display name for the group specified by groupSecurityName.
 * For this version of WebSphere Application Server, the only usage of
 * this method is by the clients (administrative console and scripting)
 * to present a descriptive name of the user if it exists.
 *
 * @param groupSecurityName the name of the group.
 * @return  the display name for the group. The display name
 *          is a registry-specific string that represents a
 *          descriptive, not necessarily unique, name for a group.
 *          If a display name does not exist return null or empty
 *          string.
 * @exception EntryNotFoundException if groupSecurityName does
 *          not exist.
 * @exception CustomRegistryException if there is any registry-
 *          specific problem
 **/
public String getGroupDisplayName(String groupSecurityName)
   throws CustomRegistryException,
          EntryNotFoundException {
   String s,displayName = null;
   BufferedReader in = null;

   if(!isValidGroup(groupSecurityName)) {
      EntryNotFoundException nsee = new EntryNotFoundException("group:"
      + groupSecurityName +  "is not valid");
      throw nsee;
   }

   try {
      in = fileOpen(GROUPFILENAME);
      while ((s=in.readLine())!=null)
      {
         if (!(s.startsWith("#") || s.trim().length() <=0 )) {
            int index = s.indexOf(":");
            int index1 = s.lastIndexOf(":");
            if ((s.substring(0,index)).equals(groupSecurityName)) {
               displayName = s.substring(index1+1);
               break;
```

```
            }
         }
      }
   } catch(Exception ex) {
      throw new CustomRegistryException(ex.getMessage(),ex);
   } finally {
      fileClose(in);
   }

   return displayName;
}

/**
 * Returns the Unique ID for a group.

 * @param     groupSecurityName the name of the group.
 * @return    The unique ID of the group. The unique ID for
 *            a group is the stringified form of some unique,
 *            registry-specific, data that serves to represent
 *            the group. For example, for the UNIX user registry,
 *            the unique ID might be the GID.
 * @exception EntryNotFoundException if groupSecurityName does
 *            not exist.
 * @exception CustomRegistryException if there is any registry-
 *            specific problem
 * @exception RemoteException as this extends java.rmi.Remote
 **/
public String getUniqueGroupId(String groupSecurityName)
   throws CustomRegistryException,
          EntryNotFoundException {
   String s,uniqueGrpId = null;
   BufferedReader in = null;
   try {
      in = fileOpen(GROUPFILENAME);
      while ((s=in.readLine())!=null)
      {
         if (!(s.startsWith("#") || s.trim().length() <=0 )) {
            int index = s.indexOf(":");
            int index1 = s.indexOf(":", index+1);
            if ((s.substring(0,index)).equals(groupSecurityName)) {
               uniqueGrpId = s.substring(index+1,index1);
               break;
            }
         }
      }
   } catch(Exception ex) {
      throw new CustomRegistryException(ex.getMessage(),ex);
   } finally {
      fileClose(in);
   }

   if (uniqueGrpId == null) {
      EntryNotFoundException nsee =
      new EntryNotFoundException("Cannot obtain the uniqueId for group:"
      + groupSecurityName);
      throw nsee;
   }

   return uniqueGrpId;
}

/**
 * Returns the Unique IDs for all the groups that contain the unique ID
```

```
 * of a user. Called during creation of a user's credential.
 *
 * @param     uniqueUserId the unique ID of the user.
 * @return    A list of all the group unique IDs that the unique user
 *            ID belongs to. The unique ID for an entry is the
 *            stringified form of some unique, registry-specific, data
 *            that serves to represent the entry.  For example, for the
 *            UNIX user registry, the unique ID for a group might be
 *            the GID and the Unique ID for the user might be the UID.
 * @exception EntryNotFoundException if uniqueUserId does not exist.
 * @exception CustomRegistryException if there is any registry-specific
 *            problem
 **/
 public List getUniqueGroupIds(String uniqueUserId)
    throws CustomRegistryException,
           EntryNotFoundException {
    String s,uniqueGrpId = null;
    BufferedReader in = null;
    List uniqueGrpIds=new ArrayList();
    try {
       in = fileOpen(USERFILENAME);
       while ((s=in.readLine())!=null)
       {
          if (!(s.startsWith("#") || s.trim().length() <=0 )) {
             int index = s.indexOf(":");
             int index1 = s.indexOf(":", index+1);
             int index2 = s.indexOf(":", index1+1);
             if ((s.substring(index1+1,index2)).equals(uniqueUserId)) {
                int lastIndex = s.lastIndexOf(":");
                String subs = s.substring(index2+1,lastIndex);
                StringTokenizer st1 = new StringTokenizer(subs, ",");
                while (st1.hasMoreTokens())
                   uniqueGrpIds.add(st1.nextToken());
                break;
             }
          }
       }
    } catch(Exception ex) {
       throw new CustomRegistryException(ex.getMessage(),ex);
    } finally {
       fileClose(in);
    }

    return uniqueGrpIds;
 }

/**
 * Returns the name for a group given its unique ID.
 *
 * @param     uniqueGroupId the unique ID of the group.
 * @return    The name of the group.
 * @exception EntryNotFoundException if the uniqueGroupId does
 *            not exist.
 * @exception CustomRegistryException if there is any registry-
 *            specific problem
 **/
 public String getGroupSecurityName(String uniqueGroupId)
    throws CustomRegistryException,
           EntryNotFoundException {
    String s,grpSecName = null;
    BufferedReader in = null;
    try {
       in = fileOpen(GROUPFILENAME);
```

```
      while ((s=in.readLine())!=null)
      {
         if (!(s.startsWith("#") || s.trim().length() <=0 )) {
            int index = s.indexOf(":");
            int index1 = s.indexOf(":", index+1);
            if ((s.substring(index+1,index1)).equals(uniqueGroupId)) {
               grpSecName = s.substring(0,index);
               break;
            }
         }
      }
   } catch (Exception ex) {
      throw new CustomRegistryException(ex.getMessage(),ex);
   } finally {
      fileClose(in);
   }

   if (grpSecName == null) {
      EntryNotFoundException ex =
         new EntryNotFoundException("Cannot obtain the group
        security name for:"  + uniqueGroupId);
      throw ex;
   }

   return grpSecName;

}

/**
 * Determines if the groupSecurityName exists in the registry
 *
 * @param     groupSecurityName the name of the group
 * @return    True if the groups exists; otherwise false
 * @exception CustomRegistryException if there is any registry-
 *            specific problem
 **/
public boolean isValidGroup(String groupSecurityName)
   throws CustomRegistryException {
   String s;
   boolean isValid = false;
   BufferedReader in = null;
   try {
      in = fileOpen(GROUPFILENAME);
      while ((s=in.readLine())!=null)
      {
         if (!(s.startsWith("#") || s.trim().length() <=0 )) {
            int index = s.indexOf(":");
            if ((s.substring(0,index)).equals(groupSecurityName)) {
               isValid=true;
               break;
            }
         }
      }
   } catch (Exception ex) {
      throw new CustomRegistryException(ex.getMessage(),ex);
   } finally {
      fileClose(in);
   }

   return isValid;
}

/**
```

```
 * Returns the securityNames of all the groups that contain the user
 *
 * This method is called by the administrative console and scripting
 * (command line) to verify that the user entered for RunAsRole mapping
 * belongs to that role in the roles to user mapping. Initially, the
 * check is done to see  if the role contains the user. If the role does
 * not contain the user explicitly, this method is called to get the groups
 * that this user belongs to so that a check can be made on the groups that
 * the role contains.
 *
 * @param     userSecurityName the name of the user
 * @return    A list of all the group securityNames that the user
 *            belongs to.
 * @exception EntryNotFoundException if user does not exist.
 * @exception CustomRegistryException if there is any registry-
 *            specific problem
 * @exception RemoteException as this extends the java.rmi.Remote
 *            interface
 **/
 public List getGroupsForUser(String userName)
    throws CustomRegistryException,
           EntryNotFoundException {
    String s;
    List grpsForUser = new ArrayList();
    BufferedReader in = null;
    try {
       in = fileOpen(GROUPFILENAME);
       while ((s=in.readLine())!=null)
       {
          if (!(s.startsWith("#") || s.trim().length() <=0 )) {
             StringTokenizer st = new StringTokenizer(s, ":");
             for (int i=0; i<2; i++)
                st.nextToken();
             String subs = st.nextToken();
             StringTokenizer st1 = new StringTokenizer(subs, ",");
             while (st1.hasMoreTokens()) {
                if((st1.nextToken()).equals(userName)) {
                   int index = s.indexOf(":");
                   grpsForUser.add(s.substring(0,index));
                }
             }
          }
       }
    } catch (Exception ex) {
       if (!isValidUser(userName)) {
          throw new EntryNotFoundException("user:"  + userName
          +  "is not valid");
       }
       throw new CustomRegistryException(ex.getMessage(),ex);
    } finally {
       fileClose(in);
    }

    return grpsForUser;
 }

/**
 * Gets a list of users in a group.
 *
 * The maximum number of users returned is defined by the
 * limit argument.
 *
 * This method is being used by the WebSphere Application Server
```

```
* Enterprise process choreographer (Enterprise) when
* staff assignments are modeled using groups.
*
* In rare situations, if you are working with a registry where
* getting all the users from any of your groups is not practical
* (for example if there are a large number of users) you can create
* the NotImplementedException for that particular group. Make sure
* that if the process choreographer is installed (or if installed later)
* the staff assignments are not modeled using these particular groups.
* If there is no concern about returning the users from groups
* in the registry it is recommended that this method be implemented
* without creating the NotImplemented exception.
* @param         groupSecurityName the name of the group
* @param         Limits the maximum number of users that should be
*                returned. This is very useful in situations where there
*                are lots of users in the registry and getting all of
*                them at once is not practical. A value of 0 implies
*                get all the users and hence must be used with care.
* @return        A Result object that contains the list of users
*                requested and a flag to indicate if more users exist.
* @deprecated    This method will be deprecated in future.
* @exception     NotImplementedException create this exception in rare
*                situations if it is not practical to get this information
*                for any of the group or groups from the registry.
* @exception     EntryNotFoundException if the group does not exist in
*                the registry
* @exception     CustomRegistryException if there is any registry-specific
*                problem
**/
public Result getUsersForGroup(String groupSecurityName, int limit)
   throws NotImplementedException,
          EntryNotFoundException,
          CustomRegistryException {
   String s, user;
   BufferedReader in = null;
   List usrsForGroup = new ArrayList();
   int count = 0;
   int newLimit = limit+1;
   Result result = new Result();

   try {
      in = fileOpen(GROUPFILENAME);
      while ((s=in.readLine())!=null)
      {
         if (!(s.startsWith("#") || s.trim().length() <=0 )) {
            int index = s.indexOf(":");
            if ((s.substring(0,index)).equals(groupSecurityName))
            {
               StringTokenizer st = new StringTokenizer(s, ":");
               for (int i=0; i<2; i++)
                  st.nextToken();
               String subs = st.nextToken();
               StringTokenizer st1 = new StringTokenizer(subs, ",");
               while (st1.hasMoreTokens()) {
                  user = st1.nextToken();
                  usrsForGroup.add(user);
                  if (limit !=0 && ++count == newLimit) {
                     usrsForGroup.remove(user);
                     result.setHasMore();
                     break;
                  }
               }
            }
```

```
            }
         }
      } catch (Exception ex) {
         if (!isValidGroup(groupSecurityName)) {
            throw new EntryNotFoundException("group:"
            + groupSecurityName
            +  "is not valid");
         }
         throw new CustomRegistryException(ex.getMessage(),ex);
      } finally {
         fileClose(in);
      }

      result.setList(usrsForGroup);
      return result;
   }

  /**
   * This method is implemented internally by the WebSphere Application Server
   * code in this release. This method is not called for the custom
   * registry implementations for this release. Return null in the
   * implementation.
   *
   **/
   public com.ibm.websphere.security.cred.WSCredential
         createCredential(String userSecurityName)
         throws CustomRegistryException,
                NotImplementedException,
                EntryNotFoundException {

      // This method is not called.
      return null;
   }

   // private methods
   private BufferedReader fileOpen(String fileName)
      throws FileNotFoundException {
      try {
         return new BufferedReader(new FileReader(fileName));
      } catch(FileNotFoundException e) {
         throw e;
      }
   }

   private void fileClose(BufferedReader in) {
      try {
         if (in != null) in.close();
      } catch(Exception e) {
         System.out.println("Error closing file" + e);
      }
   }

   private boolean match(String name, String pattern) {
      RegExpSample regexp = new RegExpSample(pattern);
      boolean matches = false;
      if(regexp.match(name))
          matches = true;
      return matches;
   }
}


//--------------------------------------------------------------------
```

```
// The program provides the Regular Expression implementation
// used in the sample for the custom user registry (FileRegistrySample).
// The pattern matching in the sample uses this program to search for the
// pattern (for users and groups).
//--------------------------------------------------------------------

class RegExpSample
{

    private boolean match(String s, int i, int j, int k)
    {
        for(; k < expr.length; k++)
label0:
            {
                Object obj = expr[k];
                if(obj == STAR)
                {
                    if(++k >= expr.length)
                        return true;
                    if(expr[k] instanceof String)
                    {
                        String s1 = (String)expr[k++];
                        int l = s1.length();
                        for(; (i = s.indexOf(s1, i)) >= 0; i++)
                            if(match(s, i + l, j, k))
                                return true;

                        return false;
                    }
                    for(; i < j; i++)
                        if(match(s, i, j, k))
                            return true;

                    return false;
                }
                if(obj == ANY)
                {
                    if(++i > j)
                        return false;
                    break label0;
                }
                if(obj instanceof char[][])
                {
                    if(i >= j)
                        return false;
                    char c = s.charAt(i++);
                    char ac[][] = (char[][])obj;
                    if(ac[0] == NOT)
                    {
                        for(int j1 = 1; j1 < ac.length; j1++)
                            if(ac[j1][0] <= c && c <= ac[j1][1])
                                return false;

                        break label0;
                    }
                    for(int k1 = 0; k1 < ac.length; k1++)
                        if(ac[k1][0] <= c && c <= ac[k1][1])
                            break label0;

                    return false;
                }
                if(obj instanceof String)
                {
```

```
                String s2 = (String)obj;
                int i1 = s2.length();
                if(!s.regionMatches(i, s2, 0, i1))
                    return false;
                i += i1;
            }
        }

    return i == j;
}

public boolean match(String s)
{
    return match(s, 0, s.length(), 0);
}

public boolean match(String s, int i, int j)
{
    return match(s, i, j, 0);
}

public RegExpSample(String s)
{
    Vector vector = new Vector();
    int i = s.length();
    StringBuffer stringbuffer = null;
    Object obj = null;
    for(int j = 0; j < i; j++)
    {
        char c = s.charAt(j);
        switch(c)
        {
        case 63: /* '?' */
            obj = ANY;
            break;

        case 42: /* '*' */
            obj = STAR;
            break;

        case 91: /* '[' */
            int k = ++j;
            Vector vector1 = new Vector();
            for(; j < i; j++)
            {
                c = s.charAt(j);
                if(j == k && c == '^')
                {
                    vector1.addElement(NOT);
                    continue;
                }
                if(c == '\\')
                {
                    if(j + 1 < i)
                        c = s.charAt(++j);
                }
                else
                if(c == ']')
                    break;
                char c1 = c;
                if(j + 2 < i && s.charAt(j + 1) == '-')
                    c1 = s.charAt(j += 2);
                char ac1[] = {
```

```
                        c, c1
                    };
                    vector1.addElement(ac1);
                }

                char ac[][] = new char[vector1.size()][];
                vector1.copyInto(ac);
                obj = ac;
                break;

            case 92: /* '\\' */
                if(j + 1 < i)
                    c = s.charAt(++j);
                break;

            }
            if(obj != null)
            {
                if(stringbuffer != null)
                {
                    vector.addElement(stringbuffer.toString());
                    stringbuffer = null;
                }
                vector.addElement(obj);
                obj = null;
            }
            else
            {
                if(stringbuffer == null)
                    stringbuffer = new StringBuffer();
                stringbuffer.append(c);
            }
        }

        if(stringbuffer != null)
            vector.addElement(stringbuffer.toString());
        expr = new Object[vector.size()];
        vector.copyInto(expr);
    }

    static final char NOT[] = new char[2];
    static final Integer ANY = new Integer(0);
    static final Integer STAR = new Integer(1);
    Object expr[];

}
```

users.props file:

This example presents the format for the `users.props` file.

Attention: The sample that is provided is intended to familiarize you with this feature. Do not use this sample in an actual production environment.

```
# 5639-D57, 5630-A36, 5630-A37, 5724-D18
# (C) COPYRIGHT International Business Machines Corp. 1997, 2005
# All Rights Reserved * Licensed Materials - Property of IBM
#
# Format:
# name:passwd:uid:gids:display name
# where name   = userId/userName of the user
#       passwd = password of the user
#       uid    = uniqueId of the user
#       gid    = groupIds of the groups that the user belongs to
#       display name = a (optional) display name for the user.
bob:bob1:123:567:bob
dave:dave1:234:678:
```

```
jay:jay1:345:678,789:Jay-Jay
ted:ted1:456:678:Teddy G
jeff:jeff1:222:789:Jeff
vikas:vikas1:333:789:vikas
bobby:bobby1:444:789:
```

groups.props file:

The following example illustrates the format for the `groups.props` file.

Attention: The sample provided is intended to familiarize you with this feature. Do not use this sample in an actual production environment.

```
# 5639-D57, 5630-A36, 5630-A37, 5724-D18
# (C) COPYRIGHT International Business Machines Corp. 1997, 2005
# All Rights Reserved * Licensed Materials - Property of IBM
#
# Format:
# name:gid:users:display name
# where name   = groupId of the group
#       gid    = uniqueId of the group
#       users  = list of all the userIds that the group contains
#       display name = a (optional) display name for the group.
admins:567:bob:Administrative group
operators:678:jay,ted,dave:Operators group
users:789:jay,jeff,vikas,bobby:
```

Developing the UserRegistry interface for using custom registries

Implementing this interface enables WebSphere Application Server security to use custom registries. This capability extends the `java.rmi` file. With a remote registry, you can complete this process remotely.

About this task

Provide implementations of the following methods.

Procedure

- Initialize the UserRegistry method, with initialize(java.util.Properties).

```
public void initialize(java.util.Properties props)
       throws CustomRegistryException,
              RemoteException;
```

  This method is called to initialize the UserRegistry method. All the properties that are defined in the Custom User Registry panel propagate to this method.

  For the `FileRegistrySample.java` sample file, the initialize method retrieves the names of the registry files that contain the user and group information.

  This method is called during server bringup to initialize the registry. This method is also called when validation is performed by the administrative console, when security is on. This method remains the same as in Version 4.x.

- Authenticate users with checkPassword(String,String).

```
public String checkPassword(String userSecurityName, String password)
      throws PasswordCheckFailedException
             CustomRegistryException,
             RemoteException;
```

  The checkPassword method is called to authenticate users when they log in using a name or user ID and a password. This method returns a string which, in most cases, is the user security name. A credential is created for the user for authorization purposes. This user name is also returned for the getCallerPrincipal enterprise bean call and the servlet calls the getUserPrincipal and getRemoteUser methods. See the getUserDisplayName method for more information if you have display names in your registry. In some situations, if you return a user other than the one who is logged in, you must verify that the user is valid in the registry.

  For the `FileRegistrySample.java` sample file, the mapCertificate method gets the distinguished name (DN) from the certificate chain and makes sure it is a valid user in the registry before returning the user.

For the sample, the checkPassword method checks the name and password combination in the user registry and, if they match, the method returns the user being authenticated.

This method is called for various scenarios, for example, by the administrative console to validate the user information after the user registry is initialized. This method is also called when you access protected resources in the product for authenticating the user and before proceeding with the authorization. This method is the same as in Version 4.x.

- Obtain user names from X.509 certificates with mapCertificate(X509Certificate[]).

```
public String mapCertificate(X509Certificate[] cert)
         throws CertificateMapNotSupportedException,
                CertificateMapFailedException,
                CustomRegistryException,
                RemoteException;
```

The mapCertificate method is called to obtain a user name from an X.509 certificate chain that is supplied by the browser. The complete certificate chain is passed to this method and the implementation can validate the chain if needed and get the user information. A credential is created for this user for authorization purposes. If browser certificates are not supported in your configuration, you can create the CertificateMapNotSupportedException exception. The consequence of not supporting certificates is authentication failure if the challenge type is certificates, even if valid certificates are in the browser.

This method is called when certificates are provided for authentication. For web applications, when the authentication constraints are set to CLIENT-CERT in the `web.xml` file of the application, this method is called to map a certificate to a valid user in the registry. For Java clients, this method is called to map the client certificates in the transport layer, when using transport layer authentication. When the identity assertion token, using the CSIv2 authentication protocol, is set to contain certificates, this method is called to map the certificates to a valid user.

In WebSphere Application Server Version 4.x, the input parameter is the X509Certificate certificate. In WebSphere Application Server Version 5.x and later, this parameter changes to accept an array of X509Certificate certificates such as a certificate chain. In Version 4.x, this parameter is called for web applications only, but in version 5.x and later, you can call this method for both web and Java clients.

- Obtain the security realm name with getRealm.

```
public String getRealm()
       throws CustomRegistryException,
       RemoteException;
```

The getRealm method is called to get the name of the security realm. The name of the realm identifies the security domain for which the registry authenticates users. If this method returns a null value, a customRealm default name is used.

For the `FileRegistrySample.java` sample file, the getRealm method returns the customRealm string. One of the calls to this method occurs when the user registry information is validated. This method is the same method as in Version 4.x.

- Obtain the list of users from the registry with getUsers(String,int).

```
public Result getUsers(String pattern, int limit)
      throws CustomRegistryException,
             RemoteException;
```

The getUsers method returns the list of users from the registry. The names of users depend on the pattern parameter. The number of users are limited by the limit parameter. In a registry that has many users, getting all the users is not practical. So the limit parameter is introduced to limit the number of users retrieved from the registry. A limit of zero (0) indicates to return all the users that match the pattern and might cause problems for large registries. Use this limit with care.

The custom registry implementations are expected to support at least the wildcard search (*). For example, a pattern of asterisk (*) returns all the users and a pattern of (b*) returns the users starting with *b*.

The return parameter is an object with a com.ibm.websphere.security.Result type. This object contains two attributes, a java.util.List and a java.lang.boolean attribute. The list contains the users that are

returned and the Boolean flag indicates if more users are available in the user registry for the search pattern. This Boolean flag is used to indicate to the client whether more users are available in the registry.

In the `FileRegistrySample.java` sample file, the getUsers method retrieves the required number of users from the user registry and sets them as a list in the Result object. To find out if more users are presented than requested, the sample gets one more user than requested and if it finds the additional user, it sets the Boolean flag to `true`. For pattern matching, the match method in the RegExpSample class is used, which supports wildcard characters such as the asterisk (*) and the question mark (?).

This method is called by the administrative console to add users to roles in the various map-users-to-roles panels. The administrative console uses the Boolean set in the Result object to indicate that more entries matching the pattern are available in the user registry.

In WebSphere Application Server Version 4.*x*, this method specifies to take only the pattern parameter. The return is a list. In WebSphere Application Server Version 5.x or later, this method is changed to take one additional parameter, the limit. Ideally, your implementation changes to take the limit value and limits the users that are returned. The return is changed to return a Result object, which consists of the list and a flag that indicates if more entries exist. When the list returns, use the Result.setList(List) method to set the list in the Result object. If more entries exist than requested in the limit parameter, set the Boolean attribute to `true` in the result object, using the Result.setHasMore method. The default for the Boolean attribute in the result object is `false`.

- Obtain the display name of a user with getUserDisplayName(String).

```
public String getUserDisplayName(String userSecurityName)
      throws EntryNotFoundException,
             CustomRegistryException,
             RemoteException;
```

The getUserDisplayName method returns a display name for a user, if one exists. The display name is an optional string that describes the user that you can set in some registries. This descriptive name is for the user and does not have to be unique in the registry.

For example in Windows systems, you can display the full name of the user.

If you do not need display names in your registry, return null or an empty string for this method.

If display names existed for any user in WebSphere Application Server Version 4.x, these names were useful for the Enterprise JavaBeans (EJB) method call getCallerPrincipal and the servlet calls getUserPrincipal and getRemoteUser. If the display names are not the same as the security name for any user, the display names are returned for the previously mentioned enterprise beans and servlet methods. Returning display names for these methods might become problematic in some situations because the display names might not be unique in the user registry. Avoid this problem by changing the default behavior to return the user security name instead of the user display name in this version of the product. For more information on how to set properties for the custom registry, see the section on *Setting Properties for Custom Registries*.

In the `FileRegistrySample.java` sample file, this method returns the display name of the user whose name matches the user name that is provided. If the display name does not exist, this method returns an empty string.

This method can be called by the product to present the display names in the administrative console, or by using the command line and the **wsadmin** tool. Use this method for display purposes only. This method is the same as in Version 4.x.

- Obtain the unique ID of a user with getUniqueUserId(String).

```
public String getUniqueUserId(String userSecurityName)
       throws EntryNotFoundException,
              CustomRegistryException,
              RemoteException;
```

This method returns the unique ID of the user, given the security name.

In the `FileRegistrySample.java` sample file, this method returns the uniqueUserId value of the user whose name matches the supplied name. This method is called when forming a credential for a user and also when creating the authorization table for the application.

- Obtain the security name of a user with getUserSecurityName(String).

```
public String getUserSecurityName(String uniqueUserId)
       throws EntryNotFoundException,
              CustomRegistryException,
              RemoteException;
```

  This method returns the security name of a user given the unique ID. In the `FileRegistrySample.java` sample file, this method returns the security name of the user whose unique ID matches the supplied ID.

  This method is called to make sure a valid user exists for a given uniqueUserId. This method is called to get the security name of the user when the uniqueUserId is obtained from a token.

- Check whether a given user is a valid user in the registry with isValidUser(String).

```
public boolean isValidUser(String userSecurityName)
        throws CustomRegistryException,
               RemoteException;
```

  This method indicates whether the given user is a valid user in the registry.

  In the `FileRegistrySample.java` sample file, this method returns `true` if the user is found in the registry, otherwise this method returns `false`. This method is primarily called in situations where knowing if the user exists in the directory prevents problems later. For example, in the mapCertificate call, when the name is obtained from the certificate if the user is not found as a valid user in the user registry, you can avoid trying to create the credential for the user.

- Return the list of groups from the user registry with getGroups(String,int).

```
public Result getGroups(String pattern, int limit)
      throws CustomRegistryException,
             RemoteException;
```

  The getGroups method returns the list of groups from the user registry. The names of groups depend on the pattern parameter. The number of groups is limited by the limit parameter. In a registry that has many groups, getting all the groups is not practical. So, the limit parameter is introduced to limit the number of groups retrieved from the user registry. A limit of zero (0) implies to return all the groups that match the pattern and can cause problems for large user registries. Use this limit with care. The custom registry implementations are expected to support at least the wildcard search (*). For example, a pattern of asterisk (*) returns all the users and a pattern of (b*) returns the users starting with *b*.

  The return parameter is an object of the com.ibm.websphere.security.Result type. This object contains the `java.util.List` and `java.lang.boolean` attributes. The list contains the groups that are returned and the Boolean flag indicates whether more groups are available in the user registry for the pattern searched. This Boolean flag is used to indicate to the client if more groups are available in the registry.

  In the `FileRegistrySample.java` sample file, the getUsers method retrieves the required number of groups from the user registry and sets them as a list in the Result object. To find out if more groups are presented than requested, the sample gets one more user than requested and if it finds the additional user, it sets the Boolean flag to `true`. For pattern matching, the match method in the RegExpSample class is used, which supports the asterisk (*) and question mark (?) characters.

  This method is called by the administrative console to add groups to roles in the various map-groups-to-roles panels. The administrative console uses the boolean set in the Result object to indicate that more entries matching the pattern are available in the user registry.

  In WebSphere Application Server Version 4, this method is used to take the pattern parameter only and returns a list. In WebSphere Application Server Version 5.x or later, this method is changed to take the limit parameter. Change to take the limit value and limit the users that are returned. The return is changed to return a Result object, which consists of the list and a flag that indicates whether more entries exist. Use the Result.setList(List) method to set the list in the Result object. If more entries exist than requested in the limit parameter, set the Boolean attribute to `true` in the Result object using the Result.setHasMore method. The default for the Boolean attribute in the Result object is `false`.

- Obtain the display name of a group with getGroupDisplayName(String).

```
public String getGroupDisplayName(String groupSecurityName)
      throws EntryNotFoundException,
             CustomRegistryException,
             RemoteException;
```

The getGroupDisplayName method returns a display name for a group if one exists. The display name is an optional string that describes the group that you can set in some user registries. This name is a descriptive name for the group and does not have to be unique in the registry. If you do not need to have display names for groups in your registry, return null or an empty string for this method.

In the `FileRegistrySample.java` sample file, this method returns the display name of the group whose name matches the group name that is provided. If the display name does not exist, this method returns an empty string.

The product can call this method to present the display names in the administrative console or through the command line using the **wsadmin** tool. This method is used for display purposes only.

- Obtain the unique ID of a group with getUniqueGroupId(String).

```
public String getUniqueGroupId(String groupSecurityName)
        throws EntryNotFoundException,
               CustomRegistryException,
               RemoteException;
```

This method returns the unique ID of the group that is given the security name.

In the `FileRegistrySample.java` sample file, this method returns the security name of the group whose unique ID matches the supplied ID. This method verifies that a valid group exists for a given uniqueGroupId ID.

- Obtain the unique IDs of all groups to which a user belongs with getUniqueGroupIds(String).

```
public List getUniqueGroupIds(String uniqueUserId)
      throws EntryNotFoundException,
             CustomRegistryException,
             RemoteException;
```

This method returns the unique IDs of all the groups to which a user belongs.

In the `FileRegistrySample.java` sample file, this method returns the unique ID of all the groups that contain this uniqueUserID ID. This method is called when creating the credential for the user. As part of creating the credential, all the groupUniqueIds IDs in which the user belongs are collected and put in the credential for authorization purposes when groups are given access to a resource.

- Obtain the security name of a group with getGroupSecurityName(String).

```
public String getGroupSecurityName(String uniqueGroupId)
     throws EntryNotFoundException,
            CustomRegistryException,
            RemoteException;
```

This method returns the security name of a group given its unique ID.

In the `FileRegistrySample.java` sample file, this method returns the security name of the group whose unique ID matches the supplied ID. This method verifies that a valid group exists for a given uniqueGroupId ID.

- Determine whether a group is a valid group in the registry with isValidGroup(String).

```
public boolean isValidGroup(String groupSecurityName)
       throws CustomRegistryException,
              RemoteException;
```

This method indicates if the given group is a valid group in the registry.

In the `FileRegistrySample.java` sample file, this method returns `true` if the group is found in the registry, otherwise the method returns `false`. This method can be used in situations where knowing whether the group exists in the directory might prevent problems later.

- Obtain all groups to which a user belongs with getGroupsForUser(String).

```
public List getGroupsForUser(String userSecurityName)
     throws EntryNotFoundException,
            CustomRegistryException,
            RemoteException;
```

This method returns all the groups to which a user belongs whose name matches the supplied name. This method is similar to the getUniqueGroupIds method with the exception that the security names are used instead of the unique IDs.

In the `FileRegistrySample.java` sample file, this method returns all the group security names that contain the userSecurityName name.

This method is called by the administrative console or the scripting tool to verify that the users entered for the RunAs roles are already part of that role in the users and groups-to-role mapping. This check is required to ensure that a user cannot be added to a RunAs role unless that user is assigned to the role in the users and groups-to-role mapping either directly or indirectly through a group that contains this user. Because a group in which the user belongs can be part of the role in the users and groups-to-role mapping, this method is called to check if any of the groups that this user belongs to mapped to that role.

- Retrieve users from a specified group with getUsersForGroup(String,int).

```
public Result getUsersForGroup(String groupSecurityName, int limit)
     throws NotImplementedException,
            EntryNotFoundException,
            CustomRegistryException,
            RemoteException;
```

This method retrieves users from the specified group. The number of users returned is limited by the limit parameter. A limit of zero (0) indicates to return all of the users in that group. This method is not directly called by the WebSphere Application Server security component. However, this method can be called by other components. In rare situations, if you are working with a user registry where getting all the users from any of your groups is not practical, you can create the NotImplementedException exception for the particular groups. In this case, verify that if the process choreographer is installed the staff assignments are not modeled using these particular groups. If no concern exists about returning the users from groups in the user registry, it is recommended that you do not create the NotImplemented exception when implementing this method.

The return parameter is an object with a com.ibm.websphere.security.Result type. This object contains the java.util.List and java.lang.boolean attributes. The list contains the users that are returned and the Boolean flag, which indicates whether more users are available in the user registry for the search pattern. This Boolean flag indicates to the client whether users are available in the user registry.

In the example, this method gets one user more than the requested number of users for a group, if the limit parameter is not set to zero (0). If the method succeeds in getting one more user, the Boolean flag is set to `true`.

In WebSphere Application Server Version 4, this getUsers method is mandatory for the product. For WebSphere Application Server Version 5.x or later, this method can create the `NotImplementedException` exception in situations where it is not practical to get the requested set of users. However, create this exception in rare situations when as other components can be affected. In Version 4, this method accepts only the pattern parameter and returns a list. In Version 5, this method accepts the limit parameter. Change your implementation to take the limit value and limit the users that are returned. The return changes to return a Result object, which consists of the list and a flag that indicates whether more entries exist. When the list is returned, use the Result.setList(List) method to set the list in the Result object. If more entries than requested are in the limit parameter, set the Boolean attribute to `true` in the Result object using Result.setHasMore method. The default for the Boolean attribute in the Result object is `false`.

- Implement the createCredential(String) method.

Attention: The first two lines of the following code sample are split for illustrative purposes only.

```
public com.ibm.websphere.security.cred.WSCredential createCredential(String userSecurityName)
 throws NotImplementedException,
            EntryNotFoundException,
            CustomRegistryException,
            RemoteException;
```

In this release the WebSphere Application Server, the createCredential method is not called. You can return *null*. In the example, a *null* value is returned.

What to do next

Managing the realm in a federated repository configuration

Follow this topic to manage the realm in a federated repository configuration.

Before you begin

The realm can consist of identities in:

- The file-based repository that is built into the system
- One or more external repositories
- Both the built-in, file-based repository and in one or more external repositories

Before you configure your realm, review "Federated repositories limitations" on page 232.

Procedure

1. Configure your realm by using one of the following topics. You might be configuring your realm for the first time or changing an existing realm configuration.
   - "Using a single built-in, file-based repository in a new configuration under Federated repositories" on page 236
   - "Changing a federated repository configuration to include a single built-in, file-based repository only" on page 246
   - "Configuring a single, Lightweight Directory Access Protocol repository in a new configuration under Federated repositories" on page 247
   - "Changing a federated repository configuration to include a single, Lightweight Directory Access Protocol repository only" on page 249
   - "Configuring multiple Lightweight Directory Access Protocol repositories in a federated repository configuration" on page 250
   - "Configuring a single built-in, file-based repository and one or more Lightweight Directory Access Protocol repositories in a federated repository configuration" on page 251
2. Configure supported entity types using the steps described in "Configuring supported entity types in a federated repository configuration" on page 295. You must configure supported entity types before you can manage this account with Users and Groups. The Base entry for the default parent determines the repository location where entities of the specified type are placed on a create operation.
3. Configure the mapping for user or group attributes of a user registry to federated repository properties in your realm using the steps described in "Configuring user repository attribute mapping in a federated repository configuration" on page 298.
4. Optional: Under Additional properties, click the **Custom properties** link to configure custom properties.
5. Optional: Use one or more of the following tasks to extend the capabilities of storing data and attributes in your realm:
   a. Configure an entry mapping repository using the steps described in "Configuring an entry mapping repository in a federated repository configuration" on page 292. An entry mapping repository is used to store data for managing profiles on multiple repositories.
   b. Configure a property extension repository using the steps described in "Configuring a property extension repository in a federated repository configuration" on page 271. A property extension repository is used to store attributes that cannot be stored in your Lightweight Directory Access Protocol (LDAP) server.
   a. Set up a database repository using wsadmin commands as described in "Setting up an entry mapping repository, a property extension repository, or a custom registry database repository using wsadmin commands" on page 277
6. Optional: Use one or more of the following advanced user tasks to extend the capabilities of LDAP repositories in your realm:

- "Increasing the performance of an LDAP repository in a federated repository configuration" on page 303
- "Configuring Lightweight Directory Access Protocol entity types in a federated repository configuration" on page 319
- "Configuring group attribute definition settings in a federated repository configuration" on page 328

7. Optional: Manage repositories that are configured in your system by following the steps described in "Managing repositories in a federated repository configuration" on page 300.
8. Optional: Add an external repository into your realm by following the steps described in "Adding an external repository in a federated repository configuration" on page 270.
9. Optional: Change the password for the repository that is configured under federated repositories by the following steps described in "Changing the password for a repository under a federated repositories configuration" on page 234.

What to do next

1. After configuring the federated repositories, click **Security > Global security** to return to the Global security panel. Verify that Federated repositories is identified in the Current realm definition field. If Federated repositories is not identified, select **Federated repositories** from the Available realm definitions field and click **Set as current**. To verify the federated repositories configuration, click **Apply** on the Global security panel. If Federated repositories is not identified in the Current realm definition field, your federated repositories configuration is not used by WebSphere Application Server.
2. If you are enabling security, complete the remaining steps as specified in "Enabling security for the realm" on page 84. As the final step, validate this setup by clicking **Apply** in the Global security panel.
3. Save, stop, and restart all the product servers (deployment managers, nodes, and Application Servers) for changes in this panel to take effect. If the server comes up without any problems, the setup is correct.

Federated repositories

Federated repositories enable you to use multiple repositories with WebSphere Application Server. These repositories, which can be file-based repositories, LDAP repositories, or a sub-tree of an LDAP repository, are defined and theoretically combined under a single realm. All of the user repositories that are configured under the federated repository functionality are invisible to WebSphere Application Server.

When you use the federated repositories functionality, all of the configured repositories, which you specify as part of the federated repository configuration, become active. It is required that the user ID, and the distinguished name (DN) for an LDAP repository, be unique in multiple user repositories that are configured under the same federated repository configuration. For example, there might be three different repositories that are configured for the federated repositories configuration: Repository A, Repository B, and Repository C. When user1 logs in, the federated repository adapter searches each of the repositories for all of the occurrences of that user. If multiple instances of that user are found in the combined repositories, an error message displays.

In addition, the federated repositories functionality in WebSphere Application Server supports the logical joining of entries across multiple user repositories when the Application Server searches and retrieves entries from the repositories. For example, when an application calls for a sorted list of people whose age is greater than twenty, WebSphere Application searches all of the repositories in the federated repositories configuration. The results are combined and sorted before the Application Server returns the results to the application.

Unlike the local operating system, stand-alone LDAP registry, or custom registry options, federated repositories provide user and group management with read and write capabilities. When you configure federated repositories, you can use one of the following methods to add, create, and delete users and groups:

Important: If you configure multiple repositories under the federated repositories realm, you must also configure supported entity types and specify a base entry for the default parent. The base

entry for the default parent determines the repository location where entities of the specified type are placed on write operations by user and group management. See “Configuring supported entity types in a federated repository configuration” on page 295 for details.

- Use the user management application programming interfaces (API). For more information, refer to articles under "Developing with virtual member manager" in this information center.
- Use the administrative console. To manage users and groups within the administrative console, click **Users and Groups > Manage Users** or **Users and Groups > Manage Groups**. For information on user and group management, click the Help link that displays in the upper right corner of the window. From the left navigation pane, click **Users and Groups**. To manage users and groups for a specific domain in a multiple security domain environment, click **Security > Global security > Security Domains > *domain_name***. Under Security Attributes, expand **User Realm**, and click **Customize for this domain**. Select the Realm type as **Federated repositories**. Click **Apply** and **Save** to the master configuration. On Security domains panel that appears, click the ***domain_name*** again to go to the domain configuration panel. Under User realm, click the **Manage users** or **Manager Groups** links that are displayed now. These links to manage users and groups for a specific domain are displayed only after you save the federated repositories configuration for the domain.
- Use the wsadmin commands. For more information, see the WIMManagementCommands command group for the AdminTask object topic.

If you do not configure the federated repositories functionality or do not enable federated repositories as the active repository, you cannot use the user management capabilities that are associated with federated repositories. You can configure an LDAP server as the active user registry and configure the same LDAP server under federated repositories, but not select federated repositories as the active user repository. With this scenario, authentication takes place using the LDAP server, and you can use the user management functionality for the LDAP server that is available for federated repositories.

The following table compares the federated repository functionality that is available in WebSphere Application Server Version 8.5 with the registry functionality that remains unchanged from previous versions of the Application Server.

Table 18. Federated repositories versus user registry implementations.

This table lists federated repositories versus user registry implementations.

Federated repositories	User registry
Supports multiple types of repositories such as file-based, LDAP, database, and custom. In WebSphere Application Server Version 8.5, file-based and LDAP repositories are supported by the administrative console. However, the federated repositories functionality does not support local operating system implementations. With this service release, the federated repositories functionality supports local operating system implementations. For database and custom repositories, you can use the wsadmin command-line interface or the configuration application programming interfaces (API).	Supports multiple types of registries such as the local operating system, a stand-alone LDAP registry, and a stand-alone custom registry.
Supports multiple repositories in a realm within a cell.	Supports one registry only in a realm within a cell.
Provides read and write capabilities for the repositories that are defined in the federated repository configuration.	Provides read only capability for the registries.
Provides account and password policy support as defined by the registry type. However, this support is not provided by the federated repository functionality.	Provides account and password policy support as defined by the registry type.
Supports identity profiles.	Does not support identity profiles.
Uses the custom UserRegistry implementation.	Uses the custom UserRegistry implementation.

Realm configuration settings

Use this page to manage the realm. The realm can consist of identities in the file-based repository that is built into the system, in one or more external repositories, or in both the built-in, file-based repository and one or more external repositories.

To view this administrative console page, complete the following steps:

1. In the administrative console, click **Security > Security domains**.
2. Under User realm, select **Customize for this domain**. Select **Federated repositories** from the Realm type field and click **Configure**.

When you finish adding or updating your federated repository configuration, go to the **Security > Global security** panel and click **Apply** to validate the changes.

A single built-in, file-based repository is built into the system and included in the realm by default.

You can configure one or more Lightweight Directory Access Protocol (LDAP) repositories to store identities in the realm. Click **Add base entry to realm** to specify a repository configuration and a base entry into the realm. You can configure multiple different base entries into the same repository.

Click **Remove** to remove selected repositories from the realm. Repository configurations and contents are not destroyed. The following restrictions apply:

- The realm must always contain at least one base entry; therefore, you cannot remove every entry.
- If you plan to remove the built-in, file-based repository from the administrative realm, verify that at least one user in another member repository is a console user with administrative rights. Otherwise, you must disable security to regain access to the administrative console.

WebSphere Application Server Version 7.0 distinguishes between the user identities for administrators who manage the environment and server identities for authenticating server to server communications. In most cases, server identities are automatically generated and are not stored in a repository.

However, if you are adding a previous version node to the latest version cell and the previous version node used a server identity and password, you must ensure that the server identity and password for the previous version are defined in the repository for this cell. Enter the server user identity and password on this panel.

Realm name:

Specifies the name of the realm. You can change the realm name.

Primary administrative user name:

Specifies the name of the user with administrative privileges that is defined in the repository, for example, adminUser.

The user name is used to log on to the administrative console when administrative security is enabled. Version 6.1 requires an administrative user that is distinct from the server user identity so that administrative actions can be audited.

Attention: In WebSphere Application Server, Version 6.0.x, a single user identity is required for both administrative access and internal process communication. When migrating to Version 6.1, this identity is used as the server user identity. You need to specify another user for the administrative user identity.

Automatically generated server identity:

Enables the application server to generate the server identity, which is recommended for environments that contain only Version 6.1 or later nodes. Automatically generated server identities are not stored in a user repository.

Information	Value
Default:	Enabled

Server identity that is stored in the repository:

Specifies a user identity in the repository that is used for internal process communication. Cells that contain Version 6.1 or later nodes require a server user identity that is defined in the active user repository.

Information	Value
Default:	Enabled

Server user ID or administrative user on a Version 6.0.x node:

Specifies the user ID that is used to run the application server for security purposes.

Password:

Specifies the password that corresponds to the server ID.

Ignore case for authorization:

Specifies that a case-insensitive authorization check is performed.

If case sensitivity is not a consideration for authorization, enable the **Ignore case for authorization** option.

Allow operations if some of the repositories are down:

Specifies whether operations (such as login, search, or get) are allowed even if the repositories in the realm are down.

Use global schema for model:

Sets the global schema option for the data model in a multiple security domain environment. Global schema refers to the schema of the admin domain.

Note: Application domains that are set to use global schema share the same schema of the admin domain. If you extend the schema for an application in one domain, you must also consider how that might affect applications of other domains, as they are bound by the same schema. For example, adding a mandatory property for one application might cause other applications to fail.

Base entry:

Specifies the base entry within the realm. This entry and its descendents are part of the realm.

Repository identifier:

Specifies a unique identifier for the repository. This identifier uniquely identifies the repository within the cell.

Repository type:

Specifies the repository type, such as File or LDAP.

User attribute mapping for federated repositories

Use this page to set or to modify the mapping for user or group attributes of a user registry to the federated repository properties in the current realm.

To view this administrative console page, click **Security > Global security**. Under Available realm definitions, click **Federated repositories**, and then **Configure**.

Note: In a multiple security domain environment, click **Security domains > *domain_name***. Under Security Attributes, expand User Realm, and click **Customize for this domain**. Select the Realm type as **Federated repositories** and then click **Configure**.

On the next page and under Additional Properties, click **User repository attribute mapping**.

Attribute mappings:

Select an attribute to set or to modify the mapping for the user or group attribute of a user registry to a federated repository property, and then click **Edit**.

Attribute
Specifies the name of the user registry attribute.

Property for Input
Specifies the name of the federated repository property that maps to the specified user registry attribute when it is an input parameter for the user registry interface.

Property for Output
Specifies the name of the federated repository property that maps to the specified user registry attribute when it is an output parameter (return value) for the user registry interface. In most cases, the propertyForInput and propertyForInput would be the same.

Custom repository details for federated repositories

Use this panel to specify the configuration for access to a custom repository.

To view this administrative console page, click **Security > Global security**. Under Available realm definitions, select **Federated repositories**, and then **Configure**. In a multiple security domain environment, click **Security domains > *domain_name***. Under Security Attributes, expand User Realm, and click **Customize for this domain**. Select the Realm type as **Federated repositories** and then click **Configure**. On the next panel, under Additional Properties, click **Manage repositories**. Under Add, select **Custom repository**.

Repository identifier:

Specifies a unique identifier for the repository. This identifier uniquely identifies the repository within the cell.

Repository adapter class name:

Specifies the implementation class name for the repository adapter. For a User Registry bridge, use com.ibm.ws.wim.adapter.urbridge.URBridge.

Login properties:

Specifies the property names to use to log into the application server.

Custom properties:

Specifies arbitrary name and value pairs of data. The name is a property key and the value is a string value that can be used to set internal system configuration properties.

Add federated repository settings

Use this page to specify the configuration for access to a file repository.

To view this administrative console page, click **Security > Global security**. Under Available realm definitions, select **Federated repositories**, and then **Configure**.

Note: In a multiple security domain environment, click **Security domains >** ***domain_name***. Under Security Attributes, expand **User Realm**, and click **Customize for this domain**. Select the Realm type as **Federated repositories** and then click **Configure**.

On the next page, under Related Items, click **Manage repositories**. Under **Add**, select **File repository**.

Repository identifier:

Specifies a unique identifier for the repository. This identifier uniquely identifies the repository within the cell.

Repository adapter class name:

Specifies the implementation class name for the repository adapter. For a User Registry bridge, use com.ibm.ws.wim.adapter.urbridge.URBridge.

Base directory:

The base directory where the files are to be created. This directory must already exist.

File name:

The file name for the repository.

The default value is `fileRegistry.xml`.

Salt length:

Specifies the salt length of the randomly generated salt for password hashing.

The default value is 12.

Message digest algorithm:

Specifies the message digest algorithm to use for hashing the password.

Select one of the following: SHA-1, SHA-256, SHA-384 or SHA-512.

The default value is SHA-1.

Login properties:

Specifies the property names to use to log into the application server.

Custom properties:

Specifies arbitrary name and value pairs of data. The name is a property key and the value is a string value that can be used to set internal system configuration properties.

Federated repositories limitations

This topic outlines known limitations and important information for configuring federated repositories.

Configuring federated repositories in a mixed-version environment

In a mixed-version deployment manager cell that contains both Version 6.1.x and Version 5.x or 6.0.x nodes, the following limitations apply for configuring federated repositories:

- You can configure only one Lightweight Directory Access Protocol (LDAP) repository under federated repositories, and the repository must be supported by Version 5.x or 6.0.x.
- You can specify a realm name that is compatible with prior versions only. The host name and the port number represent the realm for the LDAP server in a mixed-version nodes cell. For example, machine1.austin.ibm.com:389.
- You must configure a stand-alone LDAP registry; the LDAP information in both the stand-alone LDAP registry and the LDAP repository under the federated repositories configuration must match. During node synchronization, the LDAP information from the stand-alone LDAP registry propagates to the Version 5.x or 6.0.x nodes.

  Important: Before node synchronization, verify that Federated repositories is identified in the Current realm definition field. If Federated repositories is not identified, select **Federated repositories** from the Available realm definitions field and click **Set as current**. Do not set the stand-alone LDAP registry as the current realm definition.
- You cannot configure an entry mapping repository or a property extension repository in a mixed-version deployment manager cell.

Configuring LDAP servers in a federated repository

The LDAP connection `connectTimeout` default value is 20 seconds. LDAP should respond within 20 seconds for any request from WebSphere Application Server. If you cannot connect to your LDAP within this time, make sure that your LDAP is running. A connection error displays at the top of the LDAP configuration panel when the connection timeout exceeds 20 seconds.

Coexisting with Tivoli Access Manager

For Tivoli Access Manager to coexist with a federated repositories configuration, the following limitations apply:

- You can configure only one LDAP repository under federated repositories, and that LDAP repository configuration must match the LDAP server configuration under Tivoli Access Manager.
- The distinguished name for the realm base entry must match the LDAP distinguished name (DN) of the base entry within the repository. In WebSphere Application Server, Tivoli Access Manager recognizes the LDAP user ID and LDAP DN for both authentication and authorization. The federated repositories configuration does not include additional mappings for the LDAP user ID and DN.
- The federated repositories functionality does not recognize the metadata that is specified by Tivoli Access Manager. When users and groups are created under user and group management, they are not formatted using the Tivoli Access Manager metadata. The users and groups must be manually imported into Tivoli Access Manager before you use them for authentication and authorization.

Limitation for configuring active directories with their own federated repository realms

In order to use the administrative console to perform a wildcard search for all available users on two Active Directories, and to prevent multiple entries exceptions with all built-in IDs, you must first configure each Active Directory with it's own federated repository realm.

However, you cannot use the administrative console to configure each Active Directory with it's own federated repository realm. You can instead use a wsadmin script similar to the following:

```
$AdminTask createIdMgrRealm {-name AD1realm}
$AdminTask addIdMgrRealmBaseEntry {-name AD1realm -baseEntry o=AD1}

$AdminTask createIdMgrRealm {-name AD2realm}
$AdminTask addIdMgrRealmBaseEntry {-name AD2realm -baseEntry o=AD2}

$AdminConfig save
```

Limitation for repository ID in federated repositories configuration

In a federated repositories configuration, the repository ID must not exceed a length of 36 characters. If the repository ID exceeds 36 characters, an error may occur while retrieving or storing data, especially if the property extension repository is configured.

z/OS LDAP server with RACF not supported

WebSphere Application Server federated repositories DO NOT support a z/OS LDAP server with an SDBM backend (resource access control facility (RACF)).

Changing the password for a repository under a federated repositories configuration

Passwords allow security control over the repositories under a federated repositories configuration. As part of managing the realm in a federated repository configuration, one of the optional tasks you can perform is to change the password of an individual repository that is under a federated repositories configuration.

Before you begin

Before you change the password for the repository that is configured under federated repositories, ensure that the WebSphere Application Server is running and the target repository for the password change is configured under the federated repositories configuration.

Procedure

- Changing the password for a repository using the dynamic **updateIdMgrLDAPBindInfo** command Use the following steps to change the Lightweight Directory Access Protocol (LDAP) bind distinguished name (DN) or bind password of an LDAP repository.

  From a **wsadmin** prompt, you can enter the following command to display a list of arguments for the **updateIdMgrLDAPBindInfo** command: `$AdminTask help updateIdMgrLDAPBindInfo`

  1. Start the wsadmin command-line utility. The wsadmin command is found in the*app_server_root*`/bin` directory. The WebSphere Application Server and wsadmin must remaining running.
  2. Use an LDAP tool to change the password of the LDAP repository. Some LDAP repositories require a stop and start of the LDAP server to change the password.
  3. From the **wsadmin** prompt, enter the **updateIdMgrLDAPBindInfo** command to update the LDAP password under the federated repository. The change is also reflected in the wimconfig.xml file.
- Changing the password for a repository using the **updateIdMgrDBRepository** command
  1. Start the wsadmin command-line utility. The wsadmin command is found in the *app_server_root*`/bin` directory. The wsadmin command session must remain running. If WebSphere Application Server is not started, you need to open a wsadmin command session in local mode. `wsadmin -conntype none`

     gotcha: If you are starting the wsadmin command session in local mode, you must ensure that the location of the database driver is specified in the class path using the –wsadmin_classpath option. For information on using this option, see the topic, *wsadmin scripting tool* in the WebSphere Application Server information center.
  2. Log in to the Administrative Console for WebSphere Application Server.
  3. Change the password for the repository.
  4. From the Administrative Console, change the data source (J2C) password. You access the proper console page by clicking **Resources > JDBC > Data sources >***data_source***> JAAS - J2C authentication data**.
  5. From the Administrative Console, save your changes to the master configuration.
  6. From the **wsadmin** prompt, use the updateIdMgrDBRepository command to update the password in the wimconfig.xml file.

7. From the **wsadmin** prompt, save your changes to the master configuration. The following command is used to save the master configuration: `$AdminConfig save`.
8. Restart the WebSphere Application Server.

- Changing the password for a repository using the **setIdMgrPropertyExtensionRepository** command
  1. Start the wsadmin command-line utility. The wsadmin command is found in the*app_server_root*/bin directory. The wsadmin command session must remain running. If WebSphere Application Server is not started, you need to open a wsadmin command session in local mode.

     ```
     wsadmin -conntype none
     ```
  2. Log in to the Administrative Console for WebSphere Application Server.
  3. Change the password for the repository.
  4. From the Administrative Console, change the data source (J2C) password. You access the proper console page by clicking **Resources > JDBC > Data sources >** *data_source* **> JAAS - J2C authentication data**.
  5. From the Administrative Console, save your changes to the master configuration.
  6. From the **wsadmin** prompt, use the **setIdMgrPropertyExtensionRepository** command to update the password in the wimconfig.xml file.
  7. From the **wsadmin** prompt, save your changes to the master configuration. The following command is used to save the master configuration: `$AdminConfig save`.
  8. Restart the WebSphere Application Server.
- Changing the password for a repository using the **setIdMgrEntryMappingRepository** command
  1. Start the wsadmin command-line utility. The wsadmin command is found in the*app_server_root*/bin directory. The wsadmin command session must remain running. If WebSphere Application Server is not started, you need to open a wsadmin command session in local mode.

     ```
     wsadmin -conntype none
     ```
  2. Log in to the Administrative Console for WebSphere Application Server.
  3. Change the password for the repository.
  4. From the Administrative Console, change the data source (J2C) password. You access the proper console page by clicking **Resources > JDBC > Data sources >***data_source* **> JAAS - J2C authentication data**.
  5. From the Administrative Console, save your changes to the master configuration.
  6. From the **wsadmin** prompt, use the **setIdMgrEntryMappingRepository** command to update the password in the wimconfig.xml file.
  7. From the **wsadmin** prompt, save your changes to the master configuration. The following command is used to save the master configuration: `$AdminConfig save`.
  8. Restart the WebSphere Application Server.
- Changing the password for a repository using the **updateIdMgrLDAPServer** command
  1. Start the wsadmin command-line utility. The wsadmin command is found in the*app_server_root*/bin directory. The wsadmin command session must remain running. If WebSphere Application Server is not started, you need to open a wsadmin command session in local mode.

     ```
     wsadmin -conntype none
     ```
  2. Change the password for the repository.
  3. From the **wsadmin** prompt, use the **updateIdMgrLDAPServer** command to update the password in the wimconfig.xml file.
  4. From the **wsadmin** prompt, save your changes to the master configuration. The following command is used to save the master configuration: `$AdminConfig save`.
  5. Restart the WebSphere Application Server.

Results

The password for the repository has been changed.

Using a single built-in, file-based repository in a new configuration under Federated repositories

Follow this task to use a single built-in, file-based repository in a new configuration under Federated repositories.

Before you begin

To use the default configuration under Federated repositories that includes a single built-in, file-based repository only, you need to know the primary administrative user name of the user who manages WebSphere Application Server resources and user accounts.

Procedure

1. In the administrative console, click **Security > Global security**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**. To configure for a specific domain in a multiple security domain environment, click **Security domains > *domain_name***. Under Security Attributes, expand **User Realm**, and click **Customize for this domain**. Select the Realm type as **Federated repositories** and then click **Configure**.
3. Optional: Leave the Realm name field value as `defaultWIMFileBasedRealm`.
4. Enter the name of the primary administrative user in the Primary administrative user name field, for example, adminUser.
5. Leave the **Ignore case for authorization** option selected.
6. Leave the **Allow operations if some of the repositories are down** option cleared.
7. Optional: In a multiple security domain environment, select **Use global schema for model** option to indicate that the global schema option is enabled for the data model. Global schema refers to the schema of the admin domain.
8. Click **OK**.
9. Provide an administrative user password. This panel displays only when a built-in, file-based repository is included in the realm. Otherwise, the panel does not display. If a built-in, file-based repository is included, complete the following steps:
   a. Supply a password for the administrative user in the Password field.
   b. Confirm the password of the primary administrative user in the Confirm password field.
   c. Click **OK**.
10. To modify the settings of the built-in, file-based repository, under Related items, click **Manage Repositories** and then click the **InternalFileRepository** link.

    Salt length
    Specifies the salt length of the randomly generated salt for password hashing.

    Message digest algorithm
    Specifies the message digest algorithm to use for hashing the password.

    Login properties
    Specifies the property names to use to log into the application server. This field takes in multiple login properties, delimited by a semicolon (;).

    Custom properties
    Specifies arbitrary name and value pairs of data. The name is a property key and the value is a string value that can be used to set internal system configuration properties.

Results

After completing these steps, your new configuration under Federated repositories includes a single built-in, file-based repository only.

What to do next

1. Before you can manage this account with Users and Groups, configure supported entity types as described in “Configuring supported entity types in a federated repository configuration” on page 295.
2. After configuring the federated repositories, click **Security > Global security** to return to the Global security panel. Verify that Federated repositories is identified in the Current realm definition field. If Federated repositories is not identified, select **Federated repositories** from the Available realm definitions field and click **Set as current**. To verify the federated repositories configuration, click **Apply** on the Global security panel. If Federated repositories is not identified in the Current realm definition field, your federated repositories configuration is not used by WebSphere Application Server.
3. If you are enabling security, complete the remaining steps, as specified in “Enabling security for the realm” on page 84. As the final step, validate this setup by clicking **Apply** in the Global security panel.
4. Save, stop, and restart all the product servers (deployment managers, nodes, and Application Servers) for changes in this panel to take effect. If the server comes up without any problems, the setup is correct.

Administrative user password settings:

Use this page to set a password for the administrative user who manages the product resources and user accounts.

To view this administrative console page, complete the following steps:

1. In the administrative console, click **Security > Global security**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**. To configure for a specific domain in a multiple security domain environment, click **Security domains > *domain_name***. Under Security Attributes, expand User Realm, and click **Customize for this domain**. Select the Realm type as **Federated repositories** and then click **Configure**.
3. If your federated repository configuration includes a built-in, file-based repository, then the **Administrative user password** panel displays when changes are applied.

When you finish adding or updating your federated repository configuration, go to the **Security > Global security** panel and click **Apply** to validate the changes.

Password:

Specifies the password of the administrative user who manages the product resources and user accounts.

Confirm password:

Confirms the password of the administrative user who manages the product resources and user accounts.

Federated repository wizard settings:

Use this security wizard page to complete the basic requirements to connect the application server to a federated repository.

To view this security wizard page, complete the following steps

1. Click **Security > Global security > Security configuration wizard**.
2. Select your protection settings and click **Next**.
3. Select the **Federated repositories** option and click **Next**.

You can modify your federated repository configuration by completing the following steps:

1. Click **Security > Global security**.
2. Under User account repository, select Federated repository and click **Configure**.

Note: This wizard is used for the initial configuration of a built-in, file-based repository. The user name and password do not have to be in the federated repository because they will be created. If you have previously configured federated repositories, do not use the Security configuration wizard to modify your configuration. Instead, modify your configuration using the Federated repositories selection under User account repository on the Global security panel.

Primary administrative user name:

Specifies the name of the user with administrative privileges that is defined in the repository, for example, adminUser.

The user name is used to log on to the administrative console when administrative security is enabled. Version 6.1 requires an administrative user that is distinct from the server user identity so that administrative actions can be audited.

Attention: In WebSphere Application Server, Version 6.0.x, a single user identity is required for both administrative access and internal process communication. When migrating to Version 6.1, this identity is used as the server user identity. You need to specify another user for the administrative user identity.

Password:

Specifies the password of the administrative user who manages the product resources and user accounts.

Confirm password:

Confirms the password of the administrative user who manages the product resources and user accounts.

Adding a file-based repository to a federated repositories configuration

Follow this task to add a file-based repository under federated repositories.

Procedure

1. In the administrative console, click **Security > Global security**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**. To configure for a specific domain in a multiple security domain environment, click **Security domains > *domain_name***. Under Security Attributes, expand **User Realm**, and click **Customize for this domain**. Select the Realm type as **Federated repositories** and then click **Configure**.
3. Enter the name of the realm in the **Realm name** field. You can change the existing realm name.
4. Enter the name of the primary administrative user in the **Primary administrative user name** field, for example, adminUser.
5. Leave the **Ignore case for authorization** option selected.
6. Leave the **Allow operations if some of the repositories are down** option cleared.
7. Optional: In a multiple security domain environment, select **Use global schema for model** option to indicate that the global schema option is enabled for the data model. Global schema refers to the schema of the admin domain.
8. Click **Add base entry to realm**.
9. Enter a distinguished name for the realm base entry in the **Distinguished name of a base entry that uniquely identifies this set of entries in the realm** field.

10. Enter the distinguished name of the base entry within the repository in the **Distinguished name of a base entry in this repository** field.
11. Click **Add > File repository**.
12. Specify the required details for the new file repository:

    Repository identifier
    Specifies a unique identifier for the repository. This identifier uniquely identifies the repository within the cell.

    Repository adapter class name
    Specifies the implementation class name for the repository adapter. For a file repository, leave it as com.ibm.ws.wim.adapter.file.was.FileAdapter.

    Base directory
    The base directory where the files are to be created. This directory must already exist.

    File name
    The file name for the repository. The default value is fileRegistry.xml.

    Salt length
    Specifies the salt length of the randomly generated salt for password hashing. The default value is 12.

    Message digest algorithm
    Specifies the message digest algorithm to use for hashing the password. Select one of the following: SHA-1, SHA-256, SHA-384 or SHA-512. The default value is SHA-1.

    Login properties
    Specifies the property names to use to log into the application server. This field takes in multiple login properties, delimited by a semicolon (;).

    Custom properties
    Specifies arbitrary name and value pairs of data. The name is a property key and the value is a string value that can be used to set internal system configuration properties.
13. Click **OK** and **Save** to the master configuration.

Results

After completing these steps, your new configuration under Federated repositories includes a new file-based repository.

Enabling client certificate login support for a file-based repository in federated repositories

You can enable support for client certificate login in a realm configured with a single built-in file-based repository or a multiple repository configuration that includes the file-based repository and other repositories.

Before you begin

The federated repositories configuration must include a file-based repository. See the topic, Using a single built-in, file-based repository in a new configuration under Federated repositories.

About this task

The default configuration of the built-in file-based repository ignores a certificate login request, returns an empty search result, and does not display any error.

If you want to enable client certificate login for the built-in file-based repository, complete the following steps to set custom properties.

Procedure

1. In the administrative console, click **Security > Global security**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**. To configure for a specific domain in a multiple security domain environment, click **Security domains > *domain_name***. Under Security Attributes, expand **User Realm**, and click **Customize for this domain**. Select the Realm type as **Federated repositories** and then click **Configure**.
3. Under Related items, click **Manage repositories** and then click the **InternalFileRepository** link.
4. To enable certificate login for the file-based repository, under Custom properties, enter the property name as `certificateMapMode`. Specify one of the following values for this property according to your requirement:

   Note: Property names are case-sensitive, while property values are not case-sensitive.

   notSupported
   An error is displayed informing the user that the file-based repository does not support certificate login.

   exactDNMode
   Login is attempted by mapping the PrincipalName value in the X.509 certificate to the exact distinguished name (DN) in the repository. If a matching entity is found, login is successful. If a matching entity is not found, an error stating that the entity is not found is displayed.

   filterDescriptorMode
   Login is attempted using the certificate filter for the mapping. If a single matching entity is found, the login is successful. If more than one matching entity is found, the authentication fails because the result is an ambiguous match and an error is displayed.

   If you do not specify a valid value, an error is logged during initialization of the file adapter and an empty search result is returned.
5. If you set the value of the **certificateMapMode** property to `filterDescriptorMode`, then you must add another custom property, **certificateFilter**. The **certificateFilter** custom property specifies the filter that maps attributes in the client certificate to entries in the repository.

   Note: This step is not required if you set the value of the **certificateMapMode** property to `notSupported` or `exactDNMode`.

   a. Under Custom properties, click **New**.
   b. In the new row, enter the property name as `certificateFilter`. Specify the filter expression as the value for this property.

   The syntax or structure of this filter is *Repository attribute*=${*Client certificate attribute*}, for example, `uid='${SubjectCN}'`.

   The following conditions apply to the syntax of the certificate filter for file repositories:
   - The part of the filter specification that precedes the equals sign (=) must be a valid property for PersonAccount in the file-based repository.
   - The part of the filter specification that follows the equals sign (=) is one of the public attributes in your client certificate. It must begin with a dollar sign ($) and open bracket ({) and end with a close bracket (}).
   - You must enclose data for all federated repository string properties within single quotation marks (').
     For example, the federated repository property cn is a string; so a certificate filter that uses this property is specified as `cn='${IssuerCN}'`.

   You can use the following certificate attribute values on the part of the filter specification that comes after the equals sign (=). The case of the strings is important.
   - `${UniqueKey}`
   - `{PublicKey}`
   - `{PublicKey}`

- {Issuer<*xx*>} where <*xx*> is replaced by the characters that represent any valid component of the Issuer Distinguished Name. For example, you might use ${IssuerCN} for the Issuer Common Name.
- ${NotAfter}
- ${NotBefore}
- ${SerialNumber}
- ${SigAlgName}
- ${SigAlgOID}
- ${SigAlgParams}
- ${Subject<*xx*>} where <*xx*> is replaced by the characters that represent any valid component of the Subject Distinguished Name. For example, you might use ${SubjectCN} for the Subject Common Name.
- ${Version}

The following examples are complex certificate filters for the file repository:

- ((cn='${IssuerCN}') and (mobile=${SerialNumber}) and (seeAlso='${SubjectDN}'))
- ((employeeNumber=${SerialNumber}) or (seeAlso='${SubjectDN}')

There are several differences between the syntax used to specify certificate filters for LDAP repository and file repository, as shown in the following table.

Table 19. Description of differences between certificate filter syntax for LDAP and file repositories

File repository certificate filters	LDAP repository certificate filters
Use infix notations.	Use prefix notations.
Use the logical operators and and or.	Use the logical operators ampersand (&) and vertical bar (|)
Data for all federated repository string properties must be enclosed within single quotation marks ('),	Data for federated repository string properties does not have to be enclosed within single quotation marks ('),
Example: cn='${Issuer CN}' and mobile=${SerialNumber})	Example: (& (cn=${IssuerCN}) (mobile=${SerialNumber}))

6. Save the configuration changes and restart WebSphere Application Server for the changes to take effect.

Adding custom properties using wsadmin commands:

Alternately, you can also use wsadmin commands to add the custom properties as shown in the following steps.

Procedure

1. Enter the following command to start the wsadmin tool.

   ```
   wsadmin –conntype none
   ```

2. Use the setIdMgrCustomProperty command to add custom properties.

   ```
   $AdminTask setIdMgrCustomProperty { -id InternalFileRepository -name certificateMapMode -value mode}
   $AdminTask setIdMgrCustomProperty { -id InternalFileRepository -name certificateFilter -value filter_expression}
   ```

   For example, the following command searches for a user whose CN has the value specified by the IssureCN property of the certificate:

   ```
   $AdminTask setIdMgrCustomProperty { -id InternalFileRepository -name certificateFilter -value "cn='${IssuerCN}'"}
   ```

   The following command searches for a user whose CN has the value specified by the IssuerCN property of the certificate and whose mobile matches the SerialNumber property of the certificate.

   ```
   $AdminTask setIdMgrCustomProperty { -id InternalFileRepository -name certificateFilter -value "cn='${IssuerCN}' and m
   ```

3. Save the configuration changes.

   ```
   $AdminConfig save
   ```

4. Restart WebSphere Application Server for the changes to take effect.

Results

After completing these steps, support for certificate login for file-based repository is enabled in the federated repositories as shown in the following entries of the file adapter configuration:

```
<config:CustomProperties name="certificateMapMode" value="mode"/>
<config:CustomProperties name="certificateFilter" value="filter_expression"/>
```

If the certificate login request is honored, login is successful. If the certificate login request is rejected, an error is displayed.

If only file repository is configured under federated repositories, the results of the certificate login request are as described in the following table.

Table 20. Certificate login results in a federated repositories configuration that includes only a file repository

File repository	Expected results
Default behavior (**certificateMapMode** custom property is not added)	Certificate login request is ignored, an empty result is returned, and no error is displayed
Certificate login is not supported (value of **certificateMapMode** custom property is `notSupported`)	CertificateMapNotSupportedException occurs
Certificate login is supported (value of **certificateMapMode** custom property is `exactDNMode` or `filterDescriptorMode`) and user is not found	EntityNotFoundException occurs
Certificate login is supported (value of **certificateMapMode** custom property is `exactDNMode`) and an entity with DN that matches the PrincipalName in the certificate is found	Certificate login is successful
Certificate login is supported (value of **certificateMapMode** custom property is `filterDescriptorMode`) and a single matching entity is found	Certificate login is successful
Certificate login is supported (value of **certificateMapMode** custom property is `filterDescriptorMode`) and more than one matching entities are found	CertificateMapFailedException occurs and “Multiple principals found” error message is displayed

If multiple repositories are configured under federated repositories, the final login result depends on the behavior and results returned from the other repositories. The following tables contain examples of errors that are displayed in various configuration scenarios.

Table 21. Certificate login results in a federated repositories configuration that includes a file and an LDAP repository

File repository	LDAP repository	Expected results
Default behavior	Certificate login is supported and user is found	Certificate login is successful
Default behavior	Certificate login is supported and user is not found	PasswordCheckFailedException occurs
Certificate login is not supported	Certificate login is supported and user is found	CertificateMapFailedException occurs
Certificate login is supported and user is found	Certificate login is supported and user is found	DuplicateLogonIdException occurs
Certificate login is supported and user is found	Certificate login is supported and user is not found	Certificate login is successful

Table 21. Certificate login results in a federated repositories configuration that includes a file and an LDAP repository (continued)

File repository	LDAP repository	Expected results
Certificate login is supported and user is not found	Certificate login is supported and user is found	Certificate login is successful
Certificate login is supported and user is not found	Certificate login is supported and user is not found	PasswordCheckFailedException occurs

Table 22. Certificate login results in a federated repositories configuration that includes a file and local operating system repository

File repository	Local operating system repository	Expected results
Default behavior	Certificate login is not supported	CertificateMapFailedException occurs
Certificate login is not supported	Certificate login is not supported	CertificateMapNotSupportedException occurs
Certificate login is supported and user is found	Certificate login is not supported	CertificateMapFailedException occurs
Certificate login is supported and user is not found	Certificate login is not supported	CertificateMapFailedException occurs
Default behavior	Certificate login is supported and user is found	Certificate login is successful
Default behavior	Certificate login is supported and user is not found	PasswordCheckFailedException occurs
Certificate login is not supported	Certificate login is supported and user is found	CertificateMapFailedException occurs
Certificate login is supported and user is found	Certificate login is supported and user is found	DuplicateLogonIdException occurs
Certificate login is supported and user is found	Certificate login is supported and user is not found	Certificate login is successful
Certificate login is supported and user is not found	Certificate login is supported and user is found	Certificate login is successful
Certificate login is supported and user is not found	Certificate login is supported and user is not found	PasswordCheckFailedException occurs

Configuring a single built-in, file-based repository in a new configuration under federated repositories using wsadmin

You can use the Jython or Jacl scripting language with the wsadmin tool to configure a single built-in, file-based repository in a new configuration under Federated repositories.

Before you begin

Shut down the application server and ensure you have the primary administrator id and password.

About this task

The federated repositories configuration file, `wimconfig.xml`, is supported by WebSphere Application Server 6.1.x and is located in the *app_server_root*`/profiles/profile_name/config/cells/cell_name/wim/config` directory.

Use the following steps to configure for use a single built-in, file-based repository in a new configuration for federated repositories.

Procedure

1. Start the wsadmin scripting tool.
2. Create the `fileRegistry.xml` file, which is the user registry itself, if it does not already exist. If the `fileRegistry.xml` file does exist, this step just adds the user to registry.

   Using Jython:

   ```
   AdminTask.addFileRegistryAccount('-userId isoet01s01 -password oets01')
   ```

   Using Jacl:

   ```
   $AdminTask addFileRegistryAccount {-userId isoet01s01 -password oets01}
   ```

   For more information on the **addFileRegistryAccount** command, see the documentation about the FileRegistryCommands command group for the AdminTask object.
3. Update the `security.xml` file to enable administrative security, set the activeUserRegistry to use federated repositories, and update the primaryAdmin and its password.

   Using Jython:

   ```
   AdminTask applyWizardSettings('-secureApps false
    -secureLocalResources false
    -userRegistryType WIMUserRegistry
    -customRegistryClass com.ibm.ws.wim.registry.WIMUserRegistry
    -adminName isoet01s01 -adminPassword oets01')
   ```

   Using Jacl:

   ```
   $AdminTask applyWizardSettings {-secureApps false
    -secureLocalResources false
    -userRegistryType WIMUserRegistry
    -customRegistryClass com.ibm.ws.wim.registry.WIMUserRegistry
    -adminName isoet01s01
    -adminPassword oets01}
   ```

   For more information on the **applyWizardSettings** command, see the documentation about the WizardCommands command group for the AdminTask object.
4. Save your configuration changes. Enter the following commands to save the new configuration and close the wsadmin tool:

   Using Jython:

   ```
   AdminConfig.save()
   ```

   Using Jacl:

   ```
   $AdminConfig save
   ```
5. Restart the application server.

FileRegistryCommands command group for the AdminTask object:

Federated repositories provides a file registry. Use the commands in the FileRegistryCommands command group to administer the file registry using the wsadmin tool.

Note: If the Use global security settings option is selected for the user realm or the Global federated repositories option is selected as the realm type for the specified domain, the user and group management commands are executed on the federated repository of the admin domain. For example, if you run the createUser command for the specified domain, the user is created in the admin domain. However, configuration changes that are performed on the domain are applied to the security domain-specific configuration.

Use the following commands in the FileRegistryCommands group to modify the federated repository file registry:

- “addFileRegistryAccount command” on page 245
- “changeFileRegistryAccountPassword command” on page 245

addFileRegistryAccount command

The addFileRegistryAccount command adds an account to the file registry. You must save your configuration changes after running this command to save the new account to the master repository.

Target object

None

Required parameters

-userId
Specifies the ID of the user to add to the file registry. (String, required)

-password
Specifies the password of the user to add to the file registry. (String, required)

Optional parameters

-securityDomainName
Specifies the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

-parent
Specifies the parent of the entity. (String, optional)

.

Return value

This command returns a message that indicates that the command ran successfully, as the following example displays:

```
'CWWIM4544I Account newAcct(uid=newAcct,o=defaultWIMFileBasedRealm) is stored in the
file registry in the temporary workspace. You must use the "$AdminConfig save"
command to save it in the master repository.'
```

Batch mode example usage

- Using Jython string:

```
AdminTask.addFileRegistryAccount('[-userId newAcct -password new22password]')
```

Interactive mode example usage

- Using Jython string:

```
AdminTask.addFileRegistryAccount(['-userId', 'newAcct', '-password', 'new22password'])
```

changeFileRegistryAccountPassword command

The changeFileRegistryAccountPassword changes the password for the file registry account.

Target object

None.

Required parameters

-userId
Specifies the user ID of interest. (String, required)

-password
Specifies the new password. (String, required)

Optional parameters

-securityDomainName
Specifies the name that uniquely identifies the security domain. If you do not specify this parameter, the command uses the global federated repository. (String, optional)

-uniqueName
Specifies the fully-qualified unique name of the administrator. (String, optional)

Return value

This command returns a message that indicates that the command ran successfully, as the following example displays:

```
'CWWIM4545I The password is changed for newAcct(uid=newAcct,o=defaultWIMFileBasedRealm)
in the file registry in the temporary workspace. You must use the "$AdminConfig save"
command to save it in the master repository.'
```

Batch mode example usage

- Using Jython string:

```
AdminTask.changeFileRegistryAccountPassword('[-userId newAcct -password newPassword -uniqueName
uid=newAcct,o=defaultWIMFileBasedRealm]')
```

Interactive mode example usage

- Using Jython string:

```
AdminTask.changeFileRegistryAccountPassword(['-userId', 'newAcct', '-password', 'newPassword',
'-uniqueName', 'uid=newAcct,o=defaultWIMFileBasedRealm'])
```

Changing a federated repository configuration to include a single built-in, file-based repository only

Follow this task to change your federated repository configuration to include a single built-in, file-based repository only.

Before you begin

To change your federated repository configuration to include a single built-in, file-based repository only, you need to know the primary administrative user name of the user who manages WebSphere Application Server resources and user accounts.

Procedure

1. In the administrative console, click **Security > Global security**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**. To configure for a specific domain in a multiple security domain environment, click **Security domains > *domain_name***. Under Security Attributes, expand **User Realm**, and click **Customize for this domain**. Select the Realm type as **Federated repositories** and then click **Configure**.
3. Enter the name of the realm in the Realm name field. If the realm contains a single built-in, file-based repository only, you must specify `defaultWIMFileBasedRealm` as the realm name.
4. Enter the name of the primary administrative user in the Primary administrative user name field, for example, adminUser.
5. Enable the **Ignore case for authorization** option.
6. Leave the **Allow operations if some of the repositories are down** option cleared.
7. Optional: In a multiple security domain environment, select **Use global schema for model** option to indicate that the global schema option is enabled for the data model. Global schema refers to the schema of the admin domain.
8. Click **Use built-in repository** if the built-in, file-based repository is not listed in the collection.
9. Select all repositories in the collection that are not of type File and click **Remove**.

10. Click **OK**.
11. Provide an administrative user password. This panel displays only when a built-in, file-based repository is included in the realm. Otherwise, it does not display. If a built-in, file-based repository is included, complete the following steps:
    a. Supply a password for the primary administrative user in the Password field.
    b. Confirm the password of the primary administrative user in the Confirm password field.
    c. Click **OK**.
12. To modify the settings of the built-in, file-based repository, under Related items, click **Manage Repositories** and then click the **InternalFileRepository** link.

    Salt length
    Specifies the salt length of the randomly generated salt for password hashing.

    Message digest algorithm
    Specifies the message digest algorithm to use for hashing the password.

    Login properties
    Specifies the property names to use to log into the application server. This field takes in multiple login properties, delimited by a semicolon (;).

    Custom properties
    Specifies arbitrary name and value pairs of data. The name is a property key and the value is a string value that can be used to set internal system configuration properties.

Results

After completing these steps, your federated repository configuration, which includes a single built-in, file-based repository only, is configured.

What to do next

1. Before you can manage this account with Users and Groups, configure supported entity types as described in “Configuring supported entity types in a federated repository configuration” on page 295.
2. After configuring the federated repositories, click **Security > Global security** to return to the Global security panel. Verify that Federated repositories is identified in the Current realm definition field. If Federated repositories is not identified, select **Federated repositories** from the Available realm definitions field and click **Set as current**. To verify the federated repositories configuration, click **Apply** on the Global security panel. If Federated repositories is not identified in the Current realm definition field, your federated repositories configuration is not used by WebSphere Application Server.
3. If you are enabling security, complete the remaining steps, as specified in “Enabling security for the realm” on page 84. As the final step, validate this setup by clicking **Apply** in the Global security panel.
4. Save, stop, and restart all the product servers (deployment managers, nodes, and Application Servers) for changes in this panel to take effect. If the server comes up without any problems, the setup is correct.

Configuring a single, Lightweight Directory Access Protocol repository in a new configuration under Federated repositories

Follow this task to configure a single, Lightweight Directory Access Protocol (LDAP) repository in a new configuration under Federated repositories.

Before you begin

To configure an LDAP repository in a new configuration under Federated repositories, you must know a valid user name (ID), the user password, the server host and port and, if necessary, the bind distinguished name (DN) and the bind password. You can choose any valid user in the repository that is searchable. In some LDAP servers, administrative users are not searchable and cannot be used (for example, cn=root in SecureWay). This user is referred to as the WebSphere Application Server *administrative user name* or *administrative ID* in the documentation. Being an administrative ID means a user has special privileges

when calling some protected internal methods. Normally, this ID and password are used to log in to the administrative console after you turn on security. You can use other users to log in, if those users are part of the administrative roles.

Procedure

1. In the administrative console, click **Security > Global security**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**. To configure for a specific domain in a multiple security domain environment, click **Security domains > *domain_name***. Under Security Attributes, expand **User Realm**, and click **Customize for this domain**. Select the Realm type as **Federated repositories** and then click **Configure**.
3. On the Federated repositories panel, complete the following steps:
   a. Enter the name of the realm in the Realm name field. You can change the existing realm name.
   b. Enter the name of the primary administrative user in the Primary administrative user name field, for example, adminUser.
   c. Optional: Select the **Ignore case for authorization** option. When you enable this option, the authorization check is case-insensitive. Normally, an authorization check involves checking the complete DN of a user, which is unique in the realm and is case-insensitive. Clear this option when all of the member repositories in the realm are case-sensitive.

      Restriction: Some repositories contain data that is case-sensitive only, and some repositories contain data that is case-insensitive only. Do not include both case-sensitive and case-insensitive repositories in the realm. For example, do not include case-sensitive repositories in the realm with a built-in, file-based repository.
   d. Leave the **Allow operations if some of the repositories are down** option cleared.
   e. Optional: In a multiple security domain environment, select **Use global schema for model** option to indicate that the global schema option is enabled for the data model. Global schema refers to the schema of the admin domain.
   f. Click **Add base entry to realm** to add a base entry that uniquely identifies the external repository in the realm. Then complete the steps in “Adding an external repository in a federated repository configuration” on page 270.
4. On the Federated repositories panel, complete the following steps:
   a. Select the built-in, file-based repository in the collection, and click **Remove**.

      Restriction: Before you remove the built-in, file-based repository from the administrative realm, verify that at least one user in another member repository is a console user with administrative rights. Otherwise, you must disable security to regain access to the administrative console.
   b. Click **OK**.

Results

After completing these steps, your new configuration under Federated repositories includes a single, LDAP repository only.

What to do next

1. Before you can manage this account with Users and Groups, configure supported entity types as described in “Configuring supported entity types in a federated repository configuration” on page 295.
2. After configuring the federated repositories, click **Security > Global secuity** to return to the Global security panel. Verify that Federated repositories is identified in the Current realm definition field. If Federated repositories is not identified, select **Federated repositories** from the Available realm definitions field and click **Set as current**. To verify the federated repositories configuration, click **Apply**

on the Global security panel. If Federated repositories is not identified in the Current realm definition field, your federated repositories configuration is not used by WebSphere Application Server.
3. If you are enabling security, complete the remaining steps as specified in “Enabling security for the realm” on page 84. As the final step, validate this setup by clicking **Apply** in the Global security panel.
4. Save, stop, and restart all the product servers (deployment managers, nodes and Application Servers) for changes in this panel to take effect. If the server comes up without any problems, the setup is correct.

Changing a federated repository configuration to include a single, Lightweight Directory Access Protocol repository only

Follow this task to change your federated repository configuration to include a single, Lightweight Directory Access Protocol repository (LDAP) repository only.

Before you begin

To configure an LDAP repository in a federated repository configuration, you must know a valid user name (ID), the user password, the server host and port and, if necessary, the bind distinguished name (DN) and the bind password. You can choose any valid user in the repository that is searchable. In some LDAP servers, administrative users are not searchable and cannot be used (for example, cn=root in SecureWay). This user is referred to as a WebSphere Application Server *administrative user name* or *administrative ID* in the documentation. Being an administrative ID means a user has special privileges when calling some protected internal methods. Normally, this ID and password are used to log into the administrative console after you turn on security. You can use other users to log in if those users are part of the administrative roles.

Procedure

1. In the administrative console, click **Security > Global security**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**. To configure for a specific domain in a multiple security domain environment, click **Security domains > domain_name**. Under Security Attributes, expand User Realm, and click **Customize for this domain**. Select the Realm type as **Federated repositories** and then click **Configure**.
3. Enter the name of the realm in the Realm name field. You can change the existing realm name.
4. Enter the name of the primary administrative user in the Primary administrative user name field, for example, adminUser.
5. Optional: Select the **Ignore case for authorization** option. When you enable this option, the authorization check is case-insensitive. Normally, an authorization check involves checking the complete DN of a user, which is unique in the realm and is case-insensitive. Clear this option when all of the member repositories in the realm are case-sensitive.

   Restriction: Some repositories contain data that is case-sensitive only, and some repositories contain data that is case-insensitive only. Do not include both case-sensitive and case-insensitive repositories in the realm. For example, do not include case-sensitive repositories in the realm with a built-in, file-based repository.
6. Leave the **Allow operations if some of the repositories are down** option cleared.
7. Optional: In a multiple security domain environment, select **Use global schema for model** option to indicate that the global schema option is enabled for the data model. Global schema refers to the schema of the admin domain.
8. Optional: Click **Add base entry to realm** if the LDAP repository that you need is not contained in the collection. Then complete the steps in “Adding an external repository in a federated repository configuration” on page 270.
9. On the Federated repositories panel, complete the following steps:
   a. Optional: Select the repositories in the collection that you do not need in the realm and click **Remove**.

Restriction: The realm must always contain at least one base entry; therefore, you cannot remove every entry.

b. Click **OK**.

Results

After completing these steps, your federated repository configuration, which includes a single LDAP repository only, is configured.

What to do next

1. Before you can manage this account with Users and Groups, configure supported entity types as described in “Configuring supported entity types in a federated repository configuration” on page 295.
2. After configuring the federated repositories, click **Security > Global security** to return to the Global security panel. Verify that Federated repositories is identified in the Current realm definition field. If Federated repositories is not identified, select **Federated repositories** from the Available realm definitions field and click **Set as current**. To verify the federated repositories configuration, click **Apply** on the Global security panel. If Federated repositories is not identified in the Current realm definition field, your federated repositories configuration is not used by WebSphere Application Server.
3. If you are enabling security, complete the remaining steps as specified in “Enabling security for the realm” on page 84. As the final step, validate this setup by clicking **Apply** in the Global security panel.
4. Save, stop, and restart all the product servers (deployment managers, nodes, and Application Servers) for changes in this panel to take effect. If the server comes up without any problems, the setup is correct.

Configuring multiple Lightweight Directory Access Protocol repositories in a federated repository configuration

Follow this task to configure multiple Lightweight Directory Access Protocol (LDAP) repositories in a federated repository configuration.

Before you begin

To configure an LDAP repository in a federated repository configuration, you must know a valid user name (ID), the user password, the server host and port and, if necessary, the bind distinguished name (DN) and the bind password. You can choose any valid user in the repository that is searchable. In some LDAP servers, administrative users are not searchable and cannot be used (for example, cn=root in SecureWay). This user is referred to as a WebSphere Application Server *administrative user name* or *administrative ID* in the documentation. Being an administrative ID means a user has special privileges when calling some protected internal methods. Normally, this ID and password are used to log into the administrative console after you turn on security. You can use other users to log in if those users are part of the administrative roles.

Procedure

1. In the administrative console, click **Security > Global security**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**. To configure for a specific domain in a multiple security domain environment, click **Security domains > *domain_name***. Under Security Attributes, expand **User Realm**, and click **Customize for this domain**. Select the Realm type as **Federated repositories** and then click **Configure**.
3. Enter the name of the realm in the Realm name field. You can change the existing realm name.
4. Enter the name of the primary administrative user in the Primary administrative user name field, for example, adminUser.
5. Optional: Select the **Ignore case for authorization** option. When you enable this option, the authorization check is case-insensitive. Normally, an authorization check involves checking the complete DN of a user, which is unique in the realm and is case-insensitive. Clear this option when all of the member repositories in the realm are case-sensitive.

Restriction: Some repositories contain data that is case-sensitive only, and some repositories contain data that is case-insensitive only. Do not include both case-sensitive and case-insensitive repositories in the realm. For example, do not include case-sensitive repositories in the realm with a built-in, file-based repository.

6. Optional: Select the **Allow operations if some of the repositories are down** option to allow operations such get or search even if the repositories in the realm are down.
7. Optional: In a multiple security domain environment, select **Use global schema for model** option to indicate that the global schema option is enabled for the data model. Global schema refers to the schema of the admin domain.
8. Optional: Click **Add base entry to realm** if the LDAP repository that you need is not listed in the collection. Then complete the steps in "Adding an external repository in a federated repository configuration" on page 270.
9. On the Federated repositories panel, complete the following steps:
   a. Optional: Repeat step 6 if the LDAP repository that you need is not listed in the collection.
   b. Optional: Select the repositories in the collection that you do not need in the realm and click **Remove**. The following restrictions apply:
      - The realm must always contain at least one base entry; therefore, you cannot remove every entry.
      - If you plan to remove the built-in, file-based repository from the administrative realm, verify that at least one user in another member repository is a console user with administrative rights. Otherwise, you must disable security to regain access to the administrative console.
   c. Click **OK**.

Results

After completing these steps, your federated repository configuration, which includes multiple LDAP repositories, is configured.

What to do next

1. Before you can manage this account with Users and Groups, configure supported entity types as described in "Configuring supported entity types in a federated repository configuration" on page 295.
2. After configuring the federated repositories, click **Security > Global security** to return to the Global security panel. Verify that Federated repositories is identified in the Current realm definition field. If Federated repositories is not identified, select **Federated repositories** from the Available realm definitions field and click **Set as current**. To verify the federated repositories configuration, click **Apply** on the Global security panel. If Federated repositories is not identified in the Current realm definition field, your federated repositories configuration is not used by WebSphere Application Server.
3. If you are enabling security, complete the remaining steps as specified in "Enabling security for the realm" on page 84. As the final step, validate this setup by clicking **Apply** in the Global security panel.
4. Save, stop, and restart all the product servers (deployment managers, nodes, and Application Servers) for changes in this panel to take effect. If the server comes up without any problems, the setup is correct.

Configuring a single built-in, file-based repository and one or more Lightweight Directory Access Protocol repositories in a federated repository configuration

Follow this task to configure a single built-in, file-based repository and multiple Lightweight Directory Access Protocol (LDAP) repositories in a federated repository configuration.

Before you begin

To configure a built-in, file-based repository in a federated repository configuration, you must know the primary administrative user name of the user who manages WebSphere Application Server resources and user accounts.

To configure an LDAP repository in a federated repository configuration, you must know a valid user name (ID), the user password, the server host and port and, if necessary, the bind distinguished name (DN) and the bind password. You can choose any valid user in the repository that is searchable. In some LDAP servers, administrative users are not searchable and cannot be used (for example, cn=root in SecureWay). This user is referred to as a WebSphere Application Server *administrative user name* or *administrative ID* in the documentation. Being an administrative ID means a user has special privileges when calling some protected internal methods. Normally, this ID and password are used to log in to the administrative console after you turn on security. You can use other users to log in if those users are part of the administrative roles.

Procedure

1. In the administrative console, click **Security > Global security**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**. To configure for a specific domain in a multiple security domain environment, click **Security domains > *domain_name***. Under Security Attributes, expand **User Realm**, and click **Customize for this domain**. Select the Realm type as **Federated repositories** and then click **Configure**.
3. Enter the name of the realm in the Realm name field. You can change the existing realm name.
4. Enter the name of the primary administrative user in the Primary administrative user name field, for example, adminUser.

   Restriction: When you configure multiple repositories that includes a single built-in, file-based repository, the primary administrative user name must exist in the file-based repository. If the primary administrative user name does not exist in the file-based repository, then the name is created in the file-based repository. The primary administrative user name cannot exist in other repositories.
5. Select the **Ignore case for authorization** option.

   Attention: When the realm includes a built-in, file-based repository, you must enable the **Ignore case for authorization** option.
   When you enable this option, the authorization check is case-insensitive. Normally, an authorization check involves checking the complete DN of a user, which is unique in the realm and is case-insensitive. Clear this option when all of the member repositories in the realm are case-sensitive.

   Restriction: Some repositories contain data that is case-sensitive only, and some repositories contain data that is case-insensitive only. Do not include both case-sensitive and case-insensitive repositories in the realm. For example, do not include case-sensitive repositories in the realm with a built-in, file-based repository.
6. Optional: Select the **Allow operations if some of the repositories are down** option to allow operations such get or search even if the repositories in the realm are down.
7. Optional: In a multiple security domain environment, select **Use global schema for model** option to indicate that the global schema option is enabled for the data model. Global schema refers to the schema of the admin domain.
8. Optional: Click **Add base entry to realm** if the LDAP repository that you need is not contained in the collection. Then complete the steps in “Adding an external repository in a federated repository configuration” on page 270.
9. On the Federated repositories panel, complete the following steps:
   a. Optional: Repeat step 6 if the LDAP repository that you need is not listed in the collection.
   b. Click **Use built-in repository** if the built-in, file-based repository is not listed in the collection.
   c. Optional: Select the repositories in the collection that you do not need in the realm and click **Remove**.

Restriction: The realm must always contain at least one base entry; therefore, you cannot remove every entry.

d. Click **OK**.

10. Provide an administrative user password. This panel displays only when a built-in, file-based repository is included in the realm. Otherwise, the panel does not display. If a built-in, file-based repository is included, complete the following steps:
    a. Supply a password for the administrative user in the Password field.
    b. Confirm the password of the primary administrative user in the Confirm password field.
    c. Click **OK**.

Results

After completing these steps, your federated repository configuration, which includes a single built-in, file-based repository and one or more LDAP repositories, is configured.

What to do next

1. Before you can manage this account with Users and Groups, configure supported entity types as described in “Configuring supported entity types in a federated repository configuration” on page 295.
2. After configuring the federated repositories, click **Security > Global security** to return to the Global security panel. Verify that Federated repositories is identified in the Current realm definition field. If Federated repositories is not identified, select **Federated repositories** from the Available realm definitions field and click **Set as current**. To verify the federated repositories configuration, click **Apply** on the Global security panel. If Federated repositories is not identified in the Current realm definition field, your federated repositories configuration is not used by WebSphere Application Server.
3. If you are enabling security, complete the remaining steps as specified in “Enabling security for the realm” on page 84. As the final step, validate this setup by clicking **Apply** in the Global security panel.
4. Save, stop, and restart all the product servers (deployment managers, nodes, and Application Servers) for changes in this panel to take effect. If the server comes up without any problems, the setup is correct.

Manually configuring an Lightweight Directory Access Protocol repository in a federated repository configuration

Follow this topic to manually configure Lightweight Directory Access Protocol (LDAP) repository in a federated repository configuration.

Before you begin

As a prerequisite, you need to add an LDAP repository to your WebSphere Application Server configuration, where you define the following information:

Table 23. Prerequisite LDAP repository information.

This table lists prerequisite LDAP repository information,

Item Name	Example
Repository identifier	`ldaprepo1`
Directory type	IBM Tivoli Directory Server
Primary host name	`localhost`
Port	`389`
Bind distinguished name	`cn=ldapadmin`
Bind password	`yourpwd`
Login properties	`uid` (a property containing login information)

See “Lightweight Directory Access Protocol repository configuration settings” on page 260 for the specific steps you must perform to establish this LDAP repository.

About this task

At this point, you have a valid LDAP repository ready to be manually configured in a federated repository configuration.

Procedure

1. Map the federated repository entity types to the LDAP object classes.
   a. Configure the LDAP repository to match the used LDAP object class for users.
      1) In the administrative console, click **Security > Global security**.
      2) Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**. To configure for a specific domain in a multiple security domain environment, click **Security domains > *domain_name***. Under Security Attributes, expand **User Realm**, and click **Customize for this domain**. Select the Realm type as **Federated repositories** and then click **Configure**.
      3) Under Related items, click **Manage repositories**.
      4) Select the repository (for example, `ldaprepo1`).
      5) Click **LDAP entity types**.
      6) Click **PersonAccount**.
      7) Insert the *objectclass* name used in our LDAP server, for example, `inetOrgPerson`.
      8) Click **Apply**.
      9) Click **Save**.

      See “Configuring supported entity types in a federated repository configuration” on page 295 for an explanation of the supported entity types.

      See http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/topic/com.ibm.websphere.wim.doc.en/ldap.html for a description of the LDAP default mappings.
   b. Configure the LDAP repository to match the used LDAP *objectclass* for groups
      1) In the administrative console, click **Security > Global security**.
      2) Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**. To configure for a specific domain in a multiple security domain environment, click **Security domains > *domain_name***. Under Security Attributes, expand **User Realm**, and click **Customize for this domain**. Select the Realm type as **Federated repositories** and then click **Configure**.
      3) Under Related items, click **Manage repositories**.
      4) Select `ldaprepo1`.
      5) Click **LDAP entity types**.
      6) Click **Group**.
      7) Insert the objectclass name used for your LDAP server, for example, `groupOfUniqueNames`.
      8) Click **Apply**.
      9) Click **Save**.

      See “Group attribute definition settings” on page 329 for an explanation of group attribute definitions.
2. Map the federated repository property names to the LDAP attribute names.
   a. Configure the supported LDAP repository attributes.
      1) In the administrative console, click **Security > Global security**.
      2) Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**. To configure for a specific domain in a multiple security domain environment, click **Security domains > *domain_name***. Under Security Attributes, expand **User Realm**, and click **Customize for this domain**. Select the Realm type as **Federated repositories** and then click **Configure**.

3) Under Related items, click **Manage repositories** > ***repository_ID***, and then, under Additional properties, click the **LDAP attributes** link.
4) If the attribute mapping exists, you must first delete the existing mapping for the LDAP attribute, and then add a new mapping for the attribute. Select the checkbox next to the LDAP attribute name and click **Delete**.
5) To add an attribute mapping, click **Add**, and select **Supported**.
6) Enter the LDAP attribute name in the **Name** field, the federated repositories property name in the **Property name** field, and the entity type which applies the attribute mapping in the **Entity types** field.

Note: For all given federated repository properties, a one-to-one mapping is assumed. If no explicit mapping of the above type is defined, for example the federated repository property departmentNumber, the underlying LDAP attribute name, departmentNumber is assumed. See "Configuring Lightweight Directory Access Protocol attributes in a federated repository configuration" on page 323 for more information.

b. Configure the unsupported properties of the federated repository. To indicate that a given federated repository property, such as departmentNumber is not supported by any LDAP attributes, you need to define an unsupported property.
1) On the LDAP attributes panel, click **Add**, and select **Unsupported** from the drop-down menu.
2) Enter the federated repositories property name in the **Property name** field, and the entity type in the **Entity types** field.
3) Click **Apply** and then **Save**.

c. Configure the LDAP repository to match the used LDAP attributes for a user.
1) Edit the file

```
{WAS_HOME}\profiles\{profileName}\config\cells\{cellName}\wim\config\wimconfig.xml
```

2) Look for the section in this file containing the LDAP repository configuration, For example,
a)

```
<config:repositories
xsi:type="config:LdapRepositoryType"
adapterClassName="com.ibm.ws.wim.adapter.ldap.LdapAda
pter" id="ldaprepo1" ...>
```

b)

```
<config:attributeConfiguration>
```

c)

```
...
```

d)

```
<config:attributes name="anLDAPattribute"
propertyName="aVMMattribute"/>
```

e)

```
...
<config:attributeConfiguration>
```

3) Add an element of type `config:attributes` to define the mapping between a given federated depository property name, such as `departmentNumber`, to a desired LDAP attribute name, such as `warehouseSection`.

Note: For all given federated depository properties, a one-to-one mapping is assumed. If no explicit mapping of the above type is defined, for example the federated repository property `departmentNumber`, the underlying LDAP attribute name, `departmentNumber` is assumed.

d. Configure the unsupported properties of the federated repository.

To indicate that a given federated repository property, such as `departmentNumber` is not supported by any LDAP attributes, you need to define the following type of element:

```
<config:repositories xsi:type="config:LdapRepositoryType"
adapterClassName="com.ibm.ws.wim.adapter.ldap.LdapAdapter"
id="ldaprepo1" ...>
<config:attributeConfiguration>
```

```
...
<config:propertiesNotSupported name=" departmentNumber"/>
...
<config:attributeConfiguration>
```

e. Configure the LDAP repository to match the used LDAP user membership attribute in the groups.
   1) In the administrative console, click **Security > Global security**.
   2) Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**. To configure for a specific domain in a multiple security domain environment, click **Security domains > *domain_name***. Under Security Attributes, expand **User Realm**, and click **Customize for this domain**. Select the Realm type as **Federated repositories** and then click **Configure**.
   3) Under Related items, click **Manage repositories**.
   4) Select `ldaprepo1`
   5) Click **Group attribute defintions**.
   6) Click **Member attributes**.
   7) Check if your LDAP attributes (for example, `uniqueMember`) is specified for your LDAP objectclass (for example, `groupOfUniqueNames`).
      - If not specified, click **New** and add the pair (*objectclass / member attribute name*) that applies to your LDAP schema (for example, `uniqueMember / groupOfUniqueNames`
      - If specified, proceed.
   8) Click **Apply**.
   9) Click **Save**.

3. Map other LDAP settings by configuring a new base entry for the new LDAP repository.
   a. In the administrative console, click **Security > Global security**.
   b. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**. To configure for a specific domain in a multiple security domain environment, click **Security domains > *domain_name***. Under Security Attributes, expand **User Realm**, and click **Customize for this domain**. Select the Realm type as **Federated repositories** and then click **Configure**.
   c. Click **Add Base Entry to Realm**.
   d. Select `ldaprepo1`.
   e. Specifiy:
      - The base entry within the federated repository realm, for example, `o=Default Organization`
      - The base entry within the LDAP repository, for example, `o=Default Organization`
   f. Click **Apply**.
   g. Click **Save**.

   For an explanation of base entries, see the Configuring supported entity types in a federated repository configuration topic.

Results

After completing these steps, your federated repository matches the LDAP server settings.

What to do next

Configuring Lightweight Directory Access Protocol in a federated repository configuration

Follow this topic to configure Lightweight Directory Access Protocol (LDAP) settings in a federated repository configuration.

Before you begin

You have chosen among various ways to configure LDAP:

- "Configuring a single, Lightweight Directory Access Protocol repository in a new configuration under Federated repositories" on page 247
- "Changing a federated repository configuration to include a single, Lightweight Directory Access Protocol repository only" on page 249
- "Configuring multiple Lightweight Directory Access Protocol repositories in a federated repository configuration" on page 250
- "Configuring a single built-in, file-based repository and one or more Lightweight Directory Access Protocol repositories in a federated repository configuration" on page 251
- "Managing repositories in a federated repository configuration" on page 300

About this task

At this point, you are viewing the LDAP repository configuration page of the administrative console.

Procedure

1. Enter a unique identifier for the repository in the Repository identifier field. This identifier uniquely identifies the repository within the cell, for example: LDAP1.
2. Select the type of LDAP server that is used from the Directory type list. The type of LDAP server determines the default filters that are used by WebSphere Application Server.

   IBM Tivoli Directory Server users can choose either IBM Tivoli Directory Server or SecureWay as the directory type. Use the IBM Tivoli Directory Server directory type for better performance. For a list of supported LDAP servers, see "Using specific directory servers as the LDAP server" on page 185.
3. Enter the fully qualified host name of the primary LDAP server in the Primary host name field. You can enter either the IP address or the domain name system (DNS) name.
4. Enter the server port of the LDAP directory in the Port field. The host name and the port number represent the realm for this LDAP server in a mixed version nodes cell. If servers in different cells are communicating with each other using Lightweight Third Party Authentication (LTPA) tokens, these realms must match exactly in all the cells.

   The default value is 389, which is not a Secure Sockets Layer (SSL) connection. Use port 636 for a Secure Sockets Layer (SSL) connection. For some LDAP servers, you can specify a different port for a non-SSL or SSL connection. If you do not know the port to use, contact your LDAP server administrator.

   If multiple WebSphere Application Servers are installed and configured to run in the same single sign-on domain, or if WebSphere Application Server interoperates with a previous version of WebSphere Application Server, then it is important that the port number match all configurations. For example, if the LDAP port is explicitly specified as 389 in a Version 5.x or 6.0.x configuration, and WebSphere Application Server at Version 6.1 is going to interoperate with the Version 5.x or 6.0.x server, then verify that port 389 is specified explicitly for the Version 6.1 server.
5. Optional: Enter the host name of the failover LDAP server in the Failover host name field. You can specify a secondary directory server to be used in the event that your primary directory server becomes unavailable. After switching to a secondary directory server, LDAP repository attempts to reconnect to the primary directory server every 15 minutes.
6. Optional: Enter the port of the failover LDAP server in the Port field and click **Add**. The default value is 389, which is not a Secure Sockets Layer (SSL) connection. Use port 636 for a Secure Sockets Layer (SSL) connection. For some LDAP servers, you can specify a different port for a non-SSL or SSL connection. If you do not know the port to use, contact your LDAP server administrator.
7. Optional: Select the type of *referral*. A referral is an entity that is used to redirect a client request to another LDAP server. A referral contains the names and locations of other objects. It is sent by the

server to indicate that the information that the client requested can be found at another location, possibly at another server or several servers. The default value is ignore.

ignore
Referrals are ignored.

follow Referrals are followed automatically.

8. Optional: Specify the type of support for repository change tracking. The profile manager refers to this value before passing on the request to the corresponding adapter. If the value is none, then that repository is not called to retrieve the changed entities.

   none Specifies there is no change tracking support for this repository.

   native Specifies that the repository's native change tracking mechanism is used by virtual member manager to return changed entities.

9. Optional: Specify arbitrary name and value pairs of data as custom properties. The name is a property key and the value is a string value that can be used to set internal system configuration properties. Defining a new property enables you to configure a setting beyond that which is available in the administrative console.

10. Optional: Enter the bind DN name in the Bind distinguished name field, for example, cn=root. The bind DN is required if anonymous binds are not possible on the LDAP server to obtain user and group information or for write operations. In most cases, bind DN and bind password are needed. However, when anonymous bind can satisfy all of the required functions, bind DN and bind password are not needed. If the LDAP server is set up to use anonymous binds, leave this field blank. If a name is not specified, the application server binds anonymously.

    Note: To create LDAP queries or to browse, an LDAP client must bind to the LDAP server using the distinguished name (DN) of an account that has the authority to search and read the values of LDAP attributes, such as user and group information. The LDAP administrator ensures that **read access privileges** are set for the bind DN. Read access privileges allow access to the subtree of the base DN and ensure that searches of user and group information are successful.

    The directory server provides an operational attribute in each directory entry (for example, the IBM Directory Server uses ibm-entryUuid as the operational attribute). The value of this attribute is a universally unique identifier (UUID), which is chosen automatically by the directory server when the entry is added, and is expected to be unique: no other entry with the same or different name would have this same value. Directory clients may use this attribute to distinguish objects identified by a distinguished name or to locate an object after renaming. **Ensure that the bind credentials have the authority to read this attribute.**

11. Optional: Enter the password that corresponds to the bind DN in the Bind password field.

12. Optional: Enter the property names to use to log into WebSphere Application Server in the Login properties field. This field takes multiple login properties, delimited by a semicolon (;). For example, `uid;mail`.

    All login properties are searched during login. If multiple entries or no entries are found, an exception is thrown. For example, if you specify the login properties as `uid;mail` and the login ID as Bob, the search filter searches for uid=Bob or mail=Bob. When the search returns a single entry, then authentication can proceed. Otherwise, an exception is thrown.

    Note: If you define multiple login properties, the first login property is programmatically mapped to the federated repositories principalName property. For example, if you set `uid;mail` as the login properties, the LDAP attribute uid value is mapped to the federated repositories principalName property. If you define multiple login properties, after login, the first login property is returned as the value of the principalName property. For example, if you pass `joe@yourco.com` as the principalName value and the login properties are configured as uid;mail, the principalName is returned as joe.

13. Optional: Specify the LDAP attribute for Kerberos principal name. This field is enabled and can be modified only when Kerberos is configured and it is one of the active or preferred authentication mechanisms.
14. Optional: Select the certificate map mode in the Certificate mapping field. You can use the X.590 certificates for user authentication when LDAP is selected as the repository. The Certificate mapping field is used to indicate whether to map the X.509 certificates into an LDAP directory user by EXACT_DN or CERTIFICATE_FILTER. If EXACT_DN is selected, the DN in the certificate must exactly match the user entry in the LDAP server, including case and spaces.
15. If you select **CERTIFICATE_FILTER** in the Certificate mapping field, specify the LDAP filter for mapping attributes in the client certificate to entries in LDAP.

    If more than one LDAP entry matches the filter specification at run time, authentication fails because the result is an ambiguous match. The syntax or structure of this filter is:

    `LDAP attribute=${Client certificate attribute}`

    For example, `uid=${SubjectCN}`.

    The left side of the filter specification is an LDAP attribute that depends on the schema that your LDAP server is configured to use. The right side of the filter specification is one of the public attributes in your client certificate. The right side must begin with a dollar sign ($) and open bracket ({) and end with a close bracket (}). You can use the following certificate attribute values on the right side of the filter specification. The case of the strings is important:
    - `${UniqueKey}`
    - `${PublicKey}`
    - `${IssuerDN}`
    - `${Issuer<xx>}`

      where *<xx>* is replaced by the characters that represent any valid component of the Issuer Distinguished Name. For example, you might use `${IssuerCN}` for the Issuer Common Name.
    - `${NotAfter}`
    - `${NotBefore}`
    - `${SerialNumber}`
    - `${SigAlgName}`
    - `${SigAlgOID}`
    - `${SigAlgParams}`
    - `${SubjectDN}`
    - `${Subject<xx>}`

      where *<xx>* is replaced by the characters that represent any valid component of the Subject Distinguished Name. For example, you might use `${SubjectCN}` for the Subject Common Name.
    - `${Version}`
16. Optional: Select the **Require SSL communications** option if you want to use Secure Sockets Layer communications with the LDAP server.

    If you select the **Require SSL communications** option, you can select either the **Centrally managed** or **Use specific SSL alias** option.

    Centrally managed

    Enables you to specify an SSL configuration for a particular scope, such as the cell, node, server, or cluster in one location. To use the Centrally managed option, you must specify the SSL configuration for the particular set of endpoints. The Manage endpoint security configurations and trust zones panel displays all of the inbound and outbound endpoints that use the SSL protocol. If you expand the Inbound or Outbound section of the panel and click the name of a node, you can specify an SSL configuration that is used for every endpoint on that node. For an LDAP registry, you can override the inherited SSL configuration by specifying an SSL configuration for LDAP. To specify an SSL configuration for LDAP, complete the following steps:

    a. Click **Security > SSL certificate and key management > Manage endpoint security configurations and trust zones**.

b. Expand **Outbound** > *cell_name* > **Nodes** > *node_name* > **Servers** > *server_name* > **LDAP**.

Use specific SSL alias

Select the **Use specific SSL alias** option if you intend to select one of the SSL configurations in the menu that follows the option.

This configuration is used only when SSL is enabled for LDAP. The default is *DefaultSSLSettings*. To modify or create a new SSL configuration, complete the following steps:

a. Click **Security > SSL certificate and key management**.

b. Under Configuration settings, click **Manage endpoint security configurations and trust zones** > *configuration_name*.

c. Under Related items, click **SSL configurations**.

17. Click **OK**.

Results

After completing these steps, your LDAP repository settings are configured.

What to do next

Return to the appropriate task to complete the steps for your federated repository configuration:

- "Configuring a single, Lightweight Directory Access Protocol repository in a new configuration under Federated repositories" on page 247
- "Changing a federated repository configuration to include a single, Lightweight Directory Access Protocol repository only" on page 249
- "Configuring multiple Lightweight Directory Access Protocol repositories in a federated repository configuration" on page 250
- "Configuring a single built-in, file-based repository and one or more Lightweight Directory Access Protocol repositories in a federated repository configuration" on page 251
- "Managing repositories in a federated repository configuration" on page 300

Lightweight Directory Access Protocol repository configuration settings:

Use this page to configure secure access to a Lightweight Directory Access Protocol (LDAP) repository with optional failover servers.

To view this administrative console page, complete the following steps:

1. In the administrative console, click **Security > Global security**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**. To configure for a specific domain in a multiple security domain environment, click **Security domains** > ***domain_name***. Under Security Attributes, expand User Realm, and click **Customize for this domain**. Select the Realm type as **Federated repositories** and then click **Configure**.
3. Under Related items, click **Manage repositories**.
4. Click **Add** to specify a new external repository or select an external repository that is preconfigured.

When you finish adding or updating your federated repository configuration, go to the **Security > Global security** panel and click **Apply** to validate the changes.

Repository identifier:

Specifies a unique identifier for the LDAP repository. This identifier uniquely identifies the repository within the cell, for example: LDAP1.

Directory type:

Specifies the type of LDAP server to which you connect.

Expand the drop-down list to display a list of LDAP directory types.

Primary host name:

Specifies the host name of the primary LDAP server. This host name is either an IP address or a domain name service (DNS) name.

Port:

Specifies the LDAP server port.

The default value is 389, which is not a Secure Sockets Layer (SSL) connection. Use port 636 for a Secure Sockets Layer (SSL) connection. For some LDAP servers, you can specify a different port for a non-SSL or SSL connection. If you do not know the port to use, contact your LDAP server administrator.

Information	**Value**
Data type:	Integer
Default:	389
Range:	389, which is not a Secure Sockets Layer (SSL) connection 636, which is a Secure Sockets Layer (SSL) connection

Failover host name:

Specifies the host name of the failover LDAP server.

You can specify a secondary directory server to be used in the event that your primary directory server becomes unavailable. After switching to a secondary directory server, the LDAP repository attempts to reconnect to the primary directory server every 15 minutes.

Port:

Specifies the port of the failover LDAP server.

The default value is 389, which is not a Secure Sockets Layer (SSL) connection. Use port 636 for a Secure Sockets Layer (SSL) connection. For some LDAP servers, you can specify a different port for a non-SSL or SSL connection. If you do not know the port to use, contact your LDAP server administrator.

Information	**Value**
Data type:	Integer
Range:	389, which is not a Secure Sockets Layer (SSL) connection 636, which is a Secure Sockets Layer (SSL) connection

Support referrals to other LDAP servers:

Specifies how referrals that are encountered by the LDAP server are handled.

A referral is an entity that is used to redirect a client request to another LDAP server. A referral contains the names and locations of other objects. It is sent by the server to indicate that the information that the client requested can be found at another location, possibly at another server or several servers. The default value is ignore.

Information	**Value**
Default:	ignore
Range:	**ignore** Referrals are ignored. **follow** Referrals are followed automatically.

Support for repository change tracking:

Specifies the type of support for repository change tracking. The profile manager refers to this value before passing on the request to the corresponding adapter. If the value is **none**, then that repository is not called to retrieve the changed entities.

none Specifies there is no change tracking support for this repository.

native Specifies that the repository's native change tracking mechanism is used by virtual member manager to return changed entities.

Custom properties:

Specifies arbitrary name and value pairs of data. The name is a property key and the value is a string value that can be used to set internal system configuration properties.

Defining a new property enables you to configure a setting beyond that which is available in the administrative console.

Bind distinguished name:

Specifies the distinguished name (DN) for the application server to use when binding to the LDAP repository.

If no name is specified, the application server binds anonymously. In most cases, bind DN and bind password are needed. However, when anonymous bind can satisfy all of the required functions, bind DN and bind password are not needed.

Bind password:

Specifies the password for the application server to use when binding to the LDAP repository.

Login properties:

Specifies the property names to use to log into the application server.

This field takes multiple login properties, delimited by a semicolon (;). For example, `uid;mail`. All login properties are searched during login. If multiple entries or no entries are found, an exception is thrown. For example, if you specify the login properties as `uid;mail` and the login ID as Bob, the search filter searches for uid=Bob or mail=Bob. When the search returns a single entry, then authentication can proceed. Otherwise, an exception is thrown.

Note: If you define multiple login properties, the first login property is programmatically mapped to the federated repositories principalName property. For example, if you set `uid;mail` as the login properties, the LDAP attribute uid value is mapped to the federated repositories principalName property. If you define multiple login properties, after login, the first login property is returned as the

value of the principalName property. For example, if you pass `joe@yourco.com` as the principalName value and the login properties are configured as uid;mail, the principalName is returned as joe.

LDAP attribute for Kerberos principal name:

Specifies the LDAP attribute for Kerberos principal name. This field can be modified when Kerberos is configured and it is one of the active or preferred authentication mechanisms.

Certificate mapping:

Specifies whether to map X.509 certificates into an LDAP directory by EXACT_DN or CERTIFICATE_FILTER. Specify CERTIFICATE_FILTER to use the specified certificate filter for the mapping.

Certificate filter:

Specifies the filter certificate mapping property for the LDAP filter. The filter is used to map attributes in the client certificate to entries in the LDAP repository.

If more than one LDAP entry matches the filter specification at run time, authentication fails because the result is an ambiguous match. The syntax or structure of this filter is:

```
LDAP attribute=${Client certificate attribute}
```

An example of a simple certificate filter is: `uid=${SubjectCN}`.

You can also specify multiple properties and values as part of the certificate filter. Two examples of complex certificate filters are:

```
(&(cn=${IssuerCN}) (employeeNumber=${SerialNumber})

(& (issuer=${IssuerDN}) (serial=${SerialNumber}) (subjectdn=${SubjectDN}))
```

The left side of the filter specification is an LDAP attribute that depends on the schema that your LDAP server is configured to use. The right side of the filter specification is one of the public attributes in your client certificate. You can also use the **`UniqueKey`** certificate variable, which consists of the base64-encoding of the MD5 hash of the subject DN and issuer DN. The right side must begin with a dollar sign ($) and open bracket ({) and end with a close bracket (}). You can use the following certificate attribute values on the right side of the filter specification. The case of the strings is important:

- `${UniqueKey}`
- `${PublicKey}`
- `${IssuerDN}`
- `${Issuer`*xx*`}` where *xx* is replaced by the characters that represent any valid component of the Issuer Distinguished Name. For example, you might use `${IssuerCN}` for the Issuer Common Name.
- `${NotAfter}`
- `${NotBefore}`
- `${SerialNumber}`
- `${SigAlgName}`
- `${SigAlgOID}`
- `${SigAlgParams}`
- `${SubjectDN}`
- `${Subject`*xx*`}` where *xx* is replaced by the characters that represent any valid component of the Subject Distinguished Name. For example, you might use `${SubjectCN}` for the Subject Common Name.
- `${Version}`

Require SSL communications:

Specifies whether secure socket communication is enabled to the LDAP server.

When enabled, the Secure Sockets Layer (SSL) settings for LDAP are used, if specified.

Centrally managed:

Specifies that the selection of an SSL configuration is based upon the outbound topology view for the Java Naming and Directory Interface (JNDI) platform.

Centrally managed configurations support one location to maintain SSL configurations, rather than spreading them across the configuration documents.

Information	**Value**
Default:	Enabled
Range:	Enabled or Disabled

Use specific SSL alias:

Specifies the SSL configuration alias to use for LDAP outbound SSL communications.

This option overrides the centrally managed configuration for the JNDI platform.

Migrating a stand-alone LDAP repository to a federated repositories LDAP repository configuration

When configuring the security for your application server, you might need to migrate a stand-alone LDAP registry to a federated repositories LDAP repository configuration.

Before you begin

Note the specifications of your stand-alone LDAP repository that you want to migrate, for reference when configuring the LDAP repository in federated repositories. To access these fields, on the administrative console, click **Security > Global security**, and then under User account repository, select **Standalone LDAP registry** or **Federated repositories** from the Available realm definitions field and click **Configure**. To access these fields in a multiple security domain environment, click **Security > Global Security > Security domains > *domain_name***, and then, under Security Attributes, expand User Realm, and click **Customize for this domain**. Select the Realm type as **Standalone LDAP registry** or **Federated repositories**, and then click **Configure**.

The following table shows the administrative console panels and fields of the stand-alone LDAP repository configuration and their corresponding fields in a federated repositories LDAP repository configuration for mapping.

Table 24. Mapping between a stand-alone LDAP repository configuration and a federated repositories LDAP repository configuration. This table illustrates the mapping between a stand-alone LDAP repository configuration and a federated repositories LDAP repository configuration.

Stand-alone LDAP repository configuration	**LDAP repository in a federated repositories configuration**
Global security > Standalone LDAP registry General properties – Primary administrative user name	Global security > Federated repositories General properties – Primary administrative user name
Global security > Standalone LDAP registry LDAP server – Type of LDAP server	Global security > Federated repositories > Manage repositories > *repository_ID* LDAP server – Directory Type

Table 24. Mapping between a stand-alone LDAP repository configuration and a federated repositories LDAP repository configuration (continued). This table illustrates the mapping between a stand-alone LDAP repository configuration and a federated repositories LDAP repository configuration.

Stand-alone LDAP repository configuration	LDAP repository in a federated repositories configuration
Global security > Standalone LDAP registry LDAP server – Host	Global security > Federated repositories > Manage repositories > *repository_ID* LDAP server – Primary host name
Global security > Standalone LDAP registry LDAP server – Port	Global security > Federated repositories > Manage repositories > *repository_ID* LDAP server – Port
Global security > Standalone LDAP registry LDAP server – Failover hosts	Global security > Federated repositories > Manage repositories > *repository_ID* LDAP server – Failover server used when primary is not available
Global security > Standalone LDAP registry LDAP server – Base distinguished name (DN)	Global security > Federated repositories > Repository reference (Click Add Base entry to realm) General properties – Distinguished name of a base entry that uniquely identifies this set of entries in the realm and General properties – Distinguished name of a base entry in this repository
Global security > Standalone LDAP registry LDAP server – Search timeout	Global security > Federated repositories > Manage repositories > *repository_ID* > Performance General properties - Limit search time
Global security > Standalone LDAP registry LDAP server – Custom properties	Global security > Federated repositories > Custom properties
Global security > Standalone LDAP registry LDAP server – Server user identity	Global security > Federated repositories General properties – Server user identity
Global security > Standalone LDAP registry Security – Bind distinguished name (DN)	Global security > Federated repositories > Manage repositories > *repository_ID* Security – Bind distinguished name
Global security > Standalone LDAP registry Security – Bind password	Global security > Federated repositories > Manage repositories > *repository_ID* Security – Bind password
Global security > Standalone LDAP registry > Advanced Lightweight Directory Access Protocol (LDAP) user registry settings General properties – Kerberos user filter	Global security > Federated repositories > Manage repositories > *repository_ID* Security – LDAP attribute used for Kerberos principal name
Global security > Standalone LDAP registry > Advanced Lightweight Directory Access Protocol (LDAP) user registry settings General properties – Certificate map mode	Global security > Federated repositories > Manage repositories > *repository_ID* Security – Certificate mapping

Table 24. Mapping between a stand-alone LDAP repository configuration and a federated repositories LDAP repository configuration (continued). This table illustrates the mapping between a stand-alone LDAP repository configuration and a federated repositories LDAP repository configuration.

Stand-alone LDAP repository configuration	LDAP repository in a federated repositories configuration
Global security > Standalone LDAP registry > Advanced Lightweight Directory Access Protocol (LDAP) user registry settings General properties – Certificate filter	Global security > Federated repositories > Manage repositories > *repository_ID* Security – Certificate filter

The Realm name field under General Properties on the federated repositories LDAP configuration panel is not listed in the previous table because it does not have a one-to-one correspondence with a field in the stand-alone LDAP configuration panel. The host name and the port number represent the realm name for the standalone LDAP server in the WebSphere Application Server cell. For information on changing the realm name, see the topic Realm configuration settings.

The User Filter, Group Filter, User ID map, Group ID map, and Group member ID map fields also are not listed in the previous table as they do not have a one-to-one correspondence with fields in the federated repositories LDAP repository configuration panel. These LDAP attributes are set differently in the federated repositories LDAP repository configuration and involve multiple steps. These settings are explained in detail in the following sections and procedure.

About this task

Migrating from a stand-alone LDAP repository configuration to a federated repositories LDAP repository configuration involves migrating the configuration parameters, most of which are straight forward as shown in Table 1 in the previous section. Migrating the search filters is an important part of migrating a stand-alone LDAP repository configuration to a federated repository LDAP configuration; therefore, the concept and migration of LDAP search filters is described here in detail.

Stand-alone LDAP registry search filters follow the LDAP filter syntax, where you specify the attribute on which the search is based and its value.

The user filter is used for searching the registry for users. It is used to authenticate a user by using the attribute specified in the filter.

The group filter is used for searching the registry for groups. It specifies the property by which to look up groups.

Examples of commonly used LDAP user filters: In the following examples of search filters, %v is replaced with the corresponding search pattern of the user or group at run time.

```
(&(uid=%v)(objectclass=ePerson))
```

Searches for users where the uid attribute matches the specified search pattern of the ePerson object class.

```
(&(cn=%v)(objectclass=user))
```

Searches for users where the cn attribute matches the specified search pattern of the user object class.

```
(&(sAMAccountName=%v)(objectcategory=user))
```

Searches for users where the sAMAccountName attribute matches the specified search pattern of the user object category.

```
(&(userPrincipalName=%v)(objectcategory=user))
```

Searches for users where the userPrinciplalName attribute matches the specified search pattern of the user object category.

```
(&(mail=%v)(objectcategory=user))
```

Searches for users where the mail attribute matches the specified search pattern of the user object category.

```
(&(|(sAMAccountName=%v)(userPrincipalName=%v))(objectcategory=user))
```

Searches for users where the sAMAccountName or the userPrincipalName matches the specified search pattern of the user object category.

Examples of commonly used group filters:

```
(&cn=%v)(objectCategory=group)
```

Looks up groups based on their common names (cn).

```
(&(cn=%v)(|(objectclass=groupOfNames)(objectclass=groupOfUniqueNames)))
```

Looks up groups based on their common names (cn) and by using the object class of either groupOfNames or groupOfUniqueNames.

As shown in these examples, a stand-alone LDAP registry search filter consists of LDAP attributes and object classes, based on which the search or login is performed.

You can also specify the LDAP attributes and object classes in the LDAP adapter configuration of federated repositories, but they are configured differently and provide more flexibility. In federated repositories the user is represented as PersonAccount entity type and group as Group entity type. Each entity type can have its own RDN (Relative Distinguished Name) property (rdnProperties) and object class. For example, the default RDN property of PersonAccount is uid, and the default RDN property of Group is cn. The default object class mapping depends on the LDAP server type. For example, for Tivoli Directory Server, the object class for PersonAccount is inetOrgPerson and object class for Group is groupOfNames. PersonAccount can also have login properties. When a user logs in or a search is performed for a user in a user registry, these login properties are matched with the pattern. For example, if the login properties are uid and mail, then for the search pattern, a*, all the users who match uid=a* or mail=a* are returned.

gotcha: You can specify the value of User ID Map property (userIdMap) of the stand-alone LDAP repository as the RDN property (rdnProperties) or the first login property (loginProperties) in federated repositories. Though you can set both the RDN property and the login property in federated repositories, it is sufficient if you set only the RDN property. The login property is optional and you need to set it only if the login property is different from RDN property or if there are more than one login properties. If both the RDN property and login property are set, the login property takes precedence over RDN property.

Migrating search filters involves one or more of the following steps: setting the correct login properties, mapping the attributes of the back-end repository to the federated repositories properties, setting the object class, setting the search filter by using object class or object category, and setting the member or membership attribute. This mapping and configuration for federated repositories is maintained in the wimconfig.xml file.

The stand-alone LDAP registry search filter can be split into two parts:

- User or group attributes filter
- User or group object class or object category filter

For example, in the search filter, `(&(cn=%v)(objectclass=user))`:

- The attribute filter is (cn=%v)
- The object class filter is (objectclass=user)

These two filters are mapped separately in a federated repositories configuration:

- The attribute filter is mapped to the RDN properties or login properties configuration for user and to RDN properties configuration for group.
- The object class filter is mapped to the entity type configuration of the LDAP adapter.

The default attribute and object class mapping is set based on the LDAP server type but additional steps might be required to migrate these two filters:

- attribute filter:
  - Setting either or both the RDN property and login properties (if applicable)
  - Mapping the federated repository property to the LDAP attribute (if applicable)
- object class filter:
  - Setting the object class for entity type (if applicable)
  - Setting the search filter of entity type (if applicable)

Some of the steps in the following procedure include two examples. In these steps:

- Example 1 is applicable to the scenario where you are migrating the search filter `(&(cn=%v)(objectclass=ePerson))` from a stand-alone IBM Tivoli Directory Server LDAP repository to a federated repositories LDAP repository with the identifier LDAPTDS.
- Example 2 is applicable to the scenario where you are migrating the search filter `(&(|(sAMAccountName=%v)(userPrincipalName=%v))(objectcategory=user))` from a stand-alone Microsoft Active Directory LDAP repository to a federated repositories LDAP repository with the identifier LDAPAD. sAMAccountName and userPrincipalName attributes are not defined in federated repositories, so these attributes must be mapped to federated repository properties.

Procedure

1. Add the LDAP repository that you want to migrate to the federated repositories configuration.

   See Table 1 in the Before you begin section of this topic, and follow the steps described in the topic “Configuring a single, Lightweight Directory Access Protocol repository in a new configuration under Federated repositories” on page 247. These steps include links to other procedures that you must complete such as:

   - Adding an external repository in a federated repository configuration.
   - Configuring supported entity types in a federated repository configuration.
   - Configuring Lightweight Directory Access Protocol in a federated repository configuration.

   After you complete these steps, the LDAP repository that you want to migrate will be successfully configured in the federated repository configuration.
2. Set the login properties (if applicable).

   Login properties are the property names that are used to log on to the WebSphere Application Server. You can specify multiple login properties by using the semicolon (;) as a delimiter. The federated repositories properties commonly used as login properties are uid, cn, sn, givenName, mail, and so on.

   To set login properties on the administrative console, follow the steps in the topic *Lightweight Directory Access Protocol repository configuration settings*, and apply the settings under the section, Login properties.

Example 1: In the **Login properties** field, enter `cn`.

Example 2: In the **Login properties** field, enter `uid;cn`.

Complete Step 3 to map these properties to LDAP attributes.

3. Map the federated repository property to the LDAP attribute (if applicable).

   If the LDAP attribute is not a federated repository property, then the login property that you defined must be mapped to the LDAP attribute.

   a. In the administrative console, click **Security > Global security**.

   b. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**. To configure for a specific domain in a multiple security domain environment, click **Security domains > *domain_name***. Under Security Attributes, expand **User Realm**, and click **Customize for this domain**. Select the Realm type as **Federated repositories** and then click **Configure**.

   c. Under Related items, click **Manage repositories > *repository_ID***, and then, under Additional properties, click the **LDAP attributes** link.

   d. If the attribute mapping exists, you must first delete the existing mapping for the LDAP attribute, and then add a new mapping for the attribute. Select the checkbox next to the LDAP attribute name and click **Delete**.

   e. To add an attribute mapping, click **Add**, and select **Supported** from the drop-down menu. Enter the LDAP attribute name in the **Name** field, the federated repositories property name in the **Property name** field, and the entity type which applies the attribute mapping in the **Entity types** field.

   Example 1: Because the federated repository property cn is implicitly mapped to the cn LDAP attribute, no additional mapping is required.

   Example 2: Here the search filter includes two LDAP attributes, sAMAccountName and userPrincipalName.

   - For the LDAP server type, Active Directory, the LDAP attribute sAMAccountName is mapped by default to the federated repositories property, uid, as shown in the list of attributes on LDAP attributes panel. Therefore, you do not have to execute the addIdMgrLDAPAttr command to add an attribute configuration for sAMAccountName.
   - If an attribute mapping for the LDAP attribute userPrincipalName exists, then delete the existing attribute mapping before adding a new configuration.
     a. Select the checkbox next to **userPrincalName** and click **Delete**.
     b. Click **Add**, and select **Supported** from the drop-down menu.
     c. In the **Name** field, enter `userPrincipalName`.
     d. In the **Property name** field, enter `cn`.
     e. In the **Entity types** field, enter `PersonAccount`.

4. Set the object class for an entity type (if applicable).

   gotcha: Before executing this step, check the current mapping . If the object class mapping is already set, skip this step.

   To set the object class for an entity type on the administrative console, follow the steps in the topic *Lightweight Directory Access Protocol entity types settings*, and apply the following settings under the section, Object classes:

   - Specify `PersonAccount` as the entity type name for user filters
   - Specify `Group` as the entity type name for group filters.

   Example 1: In the **Entity type** field, enter `PersonAccount`.

   In the **Object classes** field, enter `ePerson`.

Example 2: In the **Entity type** field, enter `PersonAccount`.

In the **Object classes** field, enter `user`.

5. Set the search filter for the entity type (if applicable).

   Federated repositories performs the search based on the object class setting. To change this default setting and use object category as the filter, follow the steps in topic *Lightweight Directory Access Protocol entity types settings*, and apply the settings under the section, Search Filter.

   Example 1: Because the search is based on object class, no additional configuration is required.

   Example 2: In the **Search filter** field, enter `(objectcategory=user)`.

6. To migrate group filters, you must also configure the group attribute definition settings.

   The steps to configure the group attribute definition settings through the administrative console are specified in the topic *Locating user group memberships in a Lightweight Directory Access Protocol registry*, under the section, LDAP Registry within a Federated Repositories Registry. You can also use the wsadmin commands **`addIdMgrLDAPGroupDynamicMemberAttr`** or **`addIdMgrLDAPGroupMemberAttr`** that are described in the topic *IdMgrRepositoryConfig command group for the AdminTask object*.

7. Save your configuration changes
8. Restart the application server.

Results

After completing these steps, your LDAP repository is configured for use within the federated repositories configuration.

Adding an external repository in a federated repository configuration

Follow this task to add an external repository into a federated repository configuration.

Procedure

1. If the external repository that you want to add to your federated repository configuration is previously configured, select the corresponding Repository on the Repository reference panel. To access the Repository reference panel, complete the following steps:
   a. Click **Security > Global security**.
   b. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**. To configure for a specific domain in a multiple security domain environment, click **Security domains > *domain_name***. Under Security Attributes, expand **User Realm**, and click **Customize for this domain**. Select the Realm type as **Federated repositories** and then click **Configure**.
   c. Click **Add base entry to realm**.
   d. In the Repository field, select an external repository.
2. Enter a distinguished name for the realm base entry in the **Distinguished name of a base entry that uniquely identifies this set of entries in the realm** field. This base entry must uniquely identify the external repository in the realm. If multiple repositories are included in the realm, use this field to define an additional distinguished name (DN) that uniquely identifies this set of entries within the realm. For example, repositories LDAP1 and LDAP2 might both use o=ibm,c=us as the base entry in the repository. Use the DN in this field to uniquely identify this set of entries in the realm. For example: o=ibm,c=us for LDAP1 and o=ibm2,c=us for LDAP2. The specified DN in this field maps to the LDAP DN of the base entry within the repository.
3. Enter the DN of the base entry within the repository in the **Distinguished name of a base entry in this repository** field. The base entry indicates the starting point for searches in this repository. This entry and its descendents are mapped to the subtree that is identified by this unique base name entry field. For example, for a user with a DN of `cn=John Doe, ou=Rochester, o=IBM, c=US`, specify the base entry as any of the following options:

`ou=Rochester, o=IBM, c=us` or `o=IBM, c=us` or `c=us`

In most cases, this DN is the same as the distinguished name for the realm base entry.

If this field is left blank, then the subtree defaults to the root of the repository. Consult your repository administrator to determine if your repository provides support to search from the root, or create users and groups under the root without defining a suffix beforehand.

In WebSphere Application Server, the distinguished name is normalized according to the repository specification. Normalization consists of removing spaces in the base distinguished name before or after commas and equal symbols. An example of a non-normalized base distinguished name is o = ibm, c = us or o=ibm, c=us. An example of a normalized base distinguished name is o=ibm,c=us.

4. If the repository that you want to add to your realm is not previously configured, complete the following steps:
   a. Click **Add Repository** on the Repository reference panel to configure the external repository. See step 1 to access the Repository reference panel.
   b. Configure the fields on the repository configuration panel, as described in “Configuring Lightweight Directory Access Protocol in a federated repository configuration” on page 256, “Adding a file-based repository to a federated repositories configuration” on page 238, or “Adding a custom repository to a federated repositories configuration” on page 311.
   c. Select the new Repository on the Repository reference panel.
5. Click **OK**.

Results

You have added a new or previously configured external repository into your federated repository configuration.

What to do next

1. Before you can manage this account with Users and Groups, configure supported entity types as described in “Configuring supported entity types in a federated repository configuration” on page 295.
2. After configuring the federated repositories, click **Security > Global security** to return to the Global security panel. Verify that Federated repositories is identified in the Current realm definition field. If Federated repositories is not identified, select **Federated repositories** from the Available realm definitions field and click **Set as current**. To verify the federated repositories configuration, click **Apply** on the Global security panel. If Federated repositories is not identified in the Current realm definition field, your federated repositories configuration is not used by WebSphere Application Server.
3. If you are enabling security, complete the remaining steps as specified in “Enabling security for the realm” on page 84. As the final step, validate this setup by clicking **Apply** in the Global security panel.
4. Save, stop, and restart all the product servers (deployment managers, nodes and Application Servers) for changes in this panel to take effect. If the server comes up without any problems, the setup is correct.

Configuring a property extension repository in a federated repository configuration

Follow this task to configure a property extension repository to store attributes that cannot be stored in your Lightweight Directory Access Protocol (LDAP) server.

About this task

For security and business reasons, you might want to prohibit write operations to your repositories. However, applications calling the federated repository configuration might need to store additional properties for the entities. A federated repository configuration provides a *property extension repository*, which is a database regardless of the type of main profile repositories, for a propertylevel join configuration. For example, a company that uses an LDAP directory for its internal employees and a database for external customers and business partners might not allow write access to its LDAP and its database. The company can use the *property extension repository* in a federated repository configuration

to store additional properties for the people in those repositories, excluding the user ID. When an application uses the federated repository configuration to retrieve an entry for a person, the federated repository configuration transparently joins the properties of the person that is retrieved from either the LDAP or the customer's database with the properties of the person that is retrieved from the property extension repository into a single logical person entry.

When you configure a property extension repository, you can supply a valid data source, a direct connection configuration, or both. The system first tries to connect by way of the data source. If the data source is not available, then the system uses the direct access configuration.

Restriction: You cannot configure a property extension repository in a mixed version deployment manager cell.

Procedure

1. Configure the WebSphere Application Server data source. See “Configuring the WebSphere Application Server data source” on page 291.
2. If you are adding new properties (including properties that are stored in the property extension repository) to the schema, you must do the following before you create the property extension repository.
   a. Open or create the `wimxmlextension.xml` file under the *profile_root*`\config\cells\`*cell_name*`\wim\model` directory.

      Attention: Make sure the editor is on the deployment manager node.

   b. Add the schema definition of the new property. The following sample `wimxmlextension.xml` file adds a new property called ibmotherEmail to both the Person and PersonAccount entity types. This new property type is “String” and it is multiplevalued.

```
<sdo:datagraph xmlns:sdo="commonj.sdo"
    xmlns:wim="http://www.ibm.com/websphere/wim">
 <wim:schema>
  <wim:propertySchema
   nsURI="http://www.ibm.com/websphere/wim"
   dataType="String"
   multiValued="true"
   propertyName="ibm-otherEmail">
     <wim:applicableEntityTypeNames>Person
    </wim:applicableEntityTypeNames>
    <wim:applicableEntityTypeNames>PersonAccount
    </wim:applicableEntityTypeNames>
  </wim:propertySchema>
  <wim:propertySchema
   nsURI="http://www.ibm.com/websphere/wim"
   dataType="String"
   multiValued="true"
   propertyName="ibm-personalTitle">
    <wim:applicableEntityTypeNames>Person
    </wim:applicableEntityTypeNames>
    <wim:applicableEntityTypeNames>PersonAccount
    </wim:applicableEntityTypeNames>
  </wim:propertySchema>
   <wim:propertySchema
   nsURI="http://www.ibm.com/websphere/wim"
   dataType="String"
   multiValued="true"
   propertyName="ibm-middleName">
    <wim:applicableEntityTypeNames>Person
    </wim:applicableEntityTypeNames>
    <wim:applicableEntityTypeNames>PersonAccount
    </wim:applicableEntityTypeNames>
  </wim:propertySchema>
   <wim:propertySchema
   nsURI="http://www.ibm.com/websphere/wim"
   dataType="String" multiValued="true"
   propertyName="ibm-generationQualifier">
    <wim:applicableEntityTypeNames>Person
    </wim:applicableEntityTypeNames>
    <wim:applicableEntityTypeNames>PersonAccount
    </wim:applicableEntityTypeNames>
```

```
  </wim:propertySchema>
 <wim:propertySchema
  nsURI="http://www.ibm.com/websphere/wim"
  dataType="String"
  multiValued="false"
  propertyName="ibm-regionalLocale">
   <wim:applicableEntityTypeNames>Person
   </wim:applicableEntityTypeNames>
   <wim:applicableEntityTypeNames>PersonAccount
   </wim:applicableEntityTypeNames>
 </wim:propertySchema>
  <wim:propertySchema
  nsURI="http://www.ibm.com/websphere/wim"
  dataType="String"
  multiValued="false"
  propertyName="ibm-timeZone">
   <wim:applicableEntityTypeNames>Person
   </wim:applicableEntityTypeNames>
   <wim:applicableEntityTypeNames>PersonAccount
   </wim:applicableEntityTypeNames>
 </wim:propertySchema>
  <wim:propertySchema
  nsURI="http://www.ibm.com/websphere/wim"
  dataType="String"
  multiValued="false"
  propertyName="ibm-preferredCalendar">
   <wim:applicableEntityTypeNames>Person
   </wim:applicableEntityTypeNames>
   <wim:applicableEntityTypeNames>PersonAccount
   </wim:applicableEntityTypeNames>
   </wim:propertySchema>
 <wim:propertySchema
  nsURI="http://www.ibm.com/websphere/wim"
  dataType="String"
  multiValued="false"
  propertyName="ibm-alternativeCalendar">
   <wim:applicableEntityTypeNames>Person
   </wim:applicableEntityTypeNames>
   <wim:applicableEntityTypeNames>PersonAccount
   </wim:applicableEntityTypeNames>
 </wim:propertySchema>
  <wim:propertySchema
  nsURI="http://www.ibm.com/websphere/wim"
  dataType="String"
  multiValued="false"
  propertyName="ibm-firstDayOfWeek">
   <wim:applicableEntityTypeNames>Person
   </wim:applicableEntityTypeNames>
   <wim:applicableEntityTypeNames>PersonAccount
   </wim:applicableEntityTypeNames>
   </wim:propertySchema>
  <wim:propertySchema
  nsURI="http://www.ibm.com/websphere/wim"
  dataType="String"
  multiValued="false"
  propertyName="ibm-firstWorkDayOfWeek">
   <wim:applicableEntityTypeNames>Person
   </wim:applicableEntityTypeNames>
   <wim:applicableEntityTypeNames>PersonAccount
   </wim:applicableEntityTypeNames>
   </wim:propertySchema>
  <wim:propertySchema
  nsURI="http://www.ibm.com/websphere/wim"
  dataType="String"
  multiValued="false"
  propertyName="ibm-gender">
   <wim:applicableEntityTypeNames>Person
   </wim:applicableEntityTypeNames>
   <wim:applicableEntityTypeNames>PersonAccount
   </wim:applicableEntityTypeNames>
   </wim:propertySchema>
 <wim:propertySchema
  nsURI="http://www.ibm.com/websphere/wim"
  dataType="String"
  multiValued="true"
  propertyName="ibm-hobby">
   <wim:applicableEntityTypeNames>Person
   </wim:applicableEntityTypeNames>
   <wim:applicableEntityTypeNames>PersonAccount
```

```
    </wim:applicableEntityTypeNames>
  </wim:propertySchema>
 </wim:schema>
</sdo:datagraph>
```

Available data types are defined in `com.ibm.websphere.wim.SchemaConstants`. For example:

```
/**
  * Instance Class: java.lang.String
  */
 String DATA_TYPE_STRING = "String";
 /**
  * Instance Class: int
  */
 String DATA_TYPE_INT = "Int";
 /**
  * Instance Class: java.lang.Object
  */
 String DATA_TYPE_DATE = "Date";
 /**
  * Instance Class: dobjava.lang.Object
  */
 String DATA_TYPE_ANY_SIMPLE_TYPE = "AnySimpleType";
 /**
  * Instance Class: java.lang.String
  */
 String DATA_TYPE_ANY_URI = "AnyURI";
 /**
  * Instance Class: java.lang.boolean
  */
 String DATA_TYPE_BOOLEAN = "Boolean";
 /**
  * Instance Class: long
  */
 String DATA_TYPE_LONG = "Long";
 /**
  * Instance Class: double
  */
 String DATA_TYPE_DOUBLE = "Double";
 /**
  * Instance Class: short
  */
 String DATA_TYPE_SHORT = "Short";
```

c. Follow the example inside this file to define the new property definitions. The schema file for `wimlaproperties.xml` is `wimdbproperty.xsd` and is in the same directory. It can be used for reference.

3. Run the `setupIdMgrPropertyExtensionRepositoryTables` command to create the property extension repository and to add the new properties.
4. Set up the property extension repository using wsadmin by following the procedure discussed in "Setting up an entry mapping repository, a property extension repository, or a custom registry database repository using wsadmin commands" on page 277; ignore the "Before you begin" options.
5. Configure the property extension repository by completing the following steps:
   a. In the administrative console, click **Security > Global security**.
   b. Under User account repository, select **Federated repositories**, and click **Configure**. To configure for a specific domain in a multiple security domain environment, click **Security domains > *domain_name***. Under Security Attributes, expand **User Realm**, and click **Customize for this domain**. Select the Realm type as **Federated repositories** and then click **Configure**.
   c. Click **Property extension repository**.
   d. Supply the name of the data source in the Data source name field.
   e. Select the type of database that is used for the property extension repository.
   f. Supply the name of the Java database connectivity (JDBC) driver in the JDBC driver field.

      Values include:

      DB2® `com.ibm.db2.jcc.DB2Driver`

      Oracle
      `oracle.jdbc.driver.OracleDriver`

      Informix®
      `com.informix.jdbc.IfxDriver`

Microsoft SQL Server
com.microsoft.jdbc.sqlserver.SQLServerDriver

Derby org.apache.derby.jdbc.EmbeddedDriver

g. Supply the database URL that is used to access the property extension repository with JDBC in the Database URL field. Use an alphanumeric text string that conforms to the standard JDBC URL syntax.

Values include:

DB2 jdbc:db2://<*hostname*>:<*port*>/<*DB2location*>

Oracle
jdbc:oracle:thin:@<*hostname*>:<*port*>:<*dbname*>

Derby jdbc:derby:c:\derby\wim

Microsoft SQL Server
jdbc:microsoft:sqlserver://<*hostname*>:1433;databaseName=wim;selectmethod=cursor;

Informix
jdbc:informixsqli://<*hostname*>:1526/wim:INFORMIXSERVER=<*IFXServerName*>;

h. Supply the user name of the database administrator in the Database administrator user name field.

i. Supply the password of the database administrator in the Password field.

j. Specify the entity retrieval limit in the Entity retrieval limit field. The entity retrieval limit is the maximum number of entities that the system can retrieve from the property extension repository with a single database query. The default value is 200.

k. Click **OK**.

Results

After completing these steps, your federated repository configuration, which includes a property extension repository, is configured.

What to do next

1. If you are enabling security, complete the remaining steps as specified in “Enabling security for the realm” on page 84. As the final step, validate this setup by clicking **Apply** on the Global security panel.
2. Save, stop, and restart all the product servers (deployment managers, nodes, and Application Servers) for changes in this panel to take effect. If the server comes up without any problems, the setup is correct.

Property extension repository settings:

Use this page to configure a property extension repository that is used to store attributes that cannot be stored in existing repositories.

To view this administrative console page, complete the following steps:

1. In the administrative console, click **Security > Global security**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**. To configure for a specific domain in a multiple security domain environment, click **Security domains** > ***domain_name***. Under Security Attributes, expand User Realm, and click **Customize for this domain**. Select the Realm type as **Federated repositories** and then click **Configure**.
3. Under Additional properties, click **Property extension repository**.

When you finish adding or updating your federated repository configuration, go to the **Security > Global security** panel and click **Apply** to validate the changes.

Data source name:

Specifies the Java Naming and Directory Interface (JNDI) name of the data source that is used to access the property extension repository.

Information	Value
Default:	`jdbc/wimDS`

Database type:

Specifies the type of database that is used for the property extension repository.

Information	Value
Default:	`DB2`

JDBC driver:

Specifies the Java Database Connectivity (JDBC) driver that is used to access the entry mapping repository.

Values include:

DB2 `COM.ibm.db2.jcc.DB2Driver`

Oracle
`oracle.jdbc.driver.OracleDriver`

Informix
`com.informix.jdbc.IfxDriver`

DataDirect Connect
`com.ddtek.jdbc.sqlserver.SQLServerDriver`

Derby `org.apache.derby.jdbc.EmbeddedDriver`

Microsoft SQL Server
`com.microsoft.sqlserver.jdbc.SQLServerDriver`

Database URL:

Specifies the web address for the property extension repository.

Values include:

DB2 `jdbc:db2:wim`

Informix
`jdbc:informix-sqli://`*`host_name`*`:`*`port`*`/wim:INFORMIXSERVER=`*`IFXServerName`*`;`

DataDirect Connect
jdbc:datadirect:sqlserver://*host_name*:1433;databaseName=wim;selectMethod=cursor;

Derby `jdbc:derby:c:\derby\wim`

Oracle
`jdbc:oracle:thin:@`*`host_name`*`:`*`port`*`:`*`dbname`*

Microsoft SQL Server
`jdbc:sqlserver://`*`host_name`*`:1433;databaseName=wim;selectMethod=cursor;`

Database administrator user name:

Specifies the user name of the database administrator that is used to access the property extension repository.

Password:

Specifies the password that is used to enable the database administrator to access the property extension repository.

Entity retrieval limit:

Specifies the maximum number of entities that the system can retrieve from the property extension repository with a single database query.

Information	Value
Data type:	Integer
Default:	200

Setting up an entry mapping repository, a property extension repository, or a custom registry database repository using wsadmin commands:

You can set up an entry mapping repository, a property extension repository, or a custom registry database repository using wsadmin commands.

Before you begin

If you are setting up an entry mapping repository, begin with the steps described in “Configuring an entry mapping repository in a federated repository configuration” on page 292.

If you are setting up a property extension repository, begin with the steps described in “Configuring a property extension repository in a federated repository configuration” on page 271.

About this task

When you create a repository, use the appropriate wsadmin commands to define the database schema and to populate the database property definitions.

Procedure

1. Create the database. You can use any relational database product. The following examples give you tips for specific vendors.
   a. For **DB2**, open a DB2 command window or command center and enter the following:

```
db2 create database <name> using codeset UTF-8 territory US
```

   Enter the following database tuning commands:

```
db2 update database configuration for <name> using applheapsz 1024
db2 update database configuration for <name> using stmtheap 4096
db2 update database configuration for <name> using app_ctl_heap_sz 2048
db2 update database configuration for <name> using locklist 1024
db2 update database configuration for <name> using indexrec RESTART
db2 update database configuration for <name> using logfilsiz 1000
db2 update database configuration for <name> using logprimary 12
db2 update database configuration for <name> using logsecond 10
db2 update database configuration for <name> using sortheap 2048
db2set DB2_RR_TO_RS=yes
```

   b. Optional: For **Informix** databases using `dbaccess`, enter the following command:

```
CREATE DATABASE <name> WITH BUFFERED LOG
```

   c. Optional: For **Oracle** databases, the database should already exist during Oracle installation (for example, `orcl`).

2. Run the setupIdMgrEntryMappingRepositoryTables command, the setupIdMgrPropertyExtensionRepositoryTables command, or the setupIdMgrDBTables command (for custom registry repositories) by doing the following:
   a. Start WebSphere Application Server.
   b. Open a command window and go to the <WAS>/Profiles/<PROFILE_NAME>bin directory.
   c. Start wsadmin.
   d. Type the necessary commands as described below.

What to do next

Using these commands, you can:
- Specify the arguments on the command line.
- Specify the arguments in a file.

The -file option enables you to specify a file in which some or all of the parameters are specified. To use the -file argument on the command line, enter the full path to the file. Parameters in the file must be specified in key=value pairs and each must be on its own line. If a parameter is specified on both the command line and in the file, the value on the command line takes precedence.

Tips for diagnosing argument errors:
- If an argument is not properly specified on the command line or in the file, a message is returned which states that the argument was not properly specified. This might mean that the argument was not specified at all or was required for a given configuration but was not specified.
- If the argument was not specified at all, check that the parameter is specified on the command line or in the file, and that it is properly spelled and has matching case.
- If the argument was required for a given configuration but was not specified, it is possible that a value is not required solely by the command but is required for the type of database and configuration you are setting.

  For example, if you set the dn, wasAdminId, or wasAdminPassword parameters, you must also specify the dbDriver parameter.

  Additionally, if the dn, wasAdminId or wasAdminPassword parameters are specified, and the databaseType is not a Apache Derby v10.2 database, then the dbAdminId and dbAdminPassword parameters must also be specified.

The setupIdMgrDBTables command:

The setupIdMgrDBTables command creates, and populates the tables in the database that you previously created. Arguments are case-sensitive, both through the command line and the file.

Parameters:

schemaLocation (String, Required)
The location of the <WAS>/etc/wim/setup directory.

dbPropXML (String)
The location of database repository property definition XML file.

databaseType (String, Required)
The type of database. Supported databases are db2, oracle, informix, derby, sqlserver, db2zos, and db2iseries.

dbURL (String, Required)
The database URL for direct access mode. For example: jdbc:db2:wim.

dbDriver (String)
The name of the database driver. For example: com.ibm.db2.jcc.DB2Driver.

dbAdminId (String)
The database administrator ID for direct access mode. For example: db2admin.

Note: For a Apache Derby v10.2 embedded database, dbAdminId is not required.

dbAdminPassword (String)
The password associated with the dbAdminId.

Note: For a Apache Derby v10.2 embedded database, dbAdminPassword is not required.

dn (String)
The default organization uniqueName to replace. For example: `o=yourco`. If it is not set, `o=Default Organization` is used.

wasAdminId (String)
The WebSphere Application Server admin user ID. The ID should be a short name, not a uniqueName. For example: wasadmin. After creation, the uniqueName is `uid=wasadmin,<defaultOrg>`.

wasAdminPassword (String)
The WebSphere Application Server admin user password. If `wasAdminId` is set, then this parameter is mandatory.

saltLength (Integer)
The salt length of the randomly generated salt for password hashing.

encryptionKey (String)
The password encryption key. Set the password encryption key to match the encryption key in the wimconfig.xml file for the repository. If the encryption key is not set, the default is used.

derbySystemHome (String)
The home location of the Apache Derby v10.2 system if you are setting up a Apache Derby v10.2 database.

reportSqlError (String)
Specifies whether to report SQL errors while setting up databases.

file (String)
The full path to a file containing the input parameters. Each input parameter must match a corresponding parameter as it would be typed on the command line, and it must be placed in a `key=value` pair. Each pair must be on a separate line.

dbSchema (String)
The database schema where you want to create the federated repository tables. The schema should exist in the database. The default value is the default schema of the database according to the database type. Typically, the default schema is the namespace of the current database user.

The deleteIdMgrDBTables command:

The deleteIdMgrDBTables command deletes the tables in the database.

Parameters:

schemaLocation (String, Required)
The location of the `<WAS>/etc/wim/setup` directory.

databaseType (String, Required)
The type of database. Supported databases are db2, oracle, informix, derby, sqlserver, db2zos, and db2iseries.

dbURL (String, Required)
The database URL for direct access mode. For example: jdbc:db2:wim.

dbDriver (String)
The name of the database driver. For example: com.ibm.db2.jcc.DB2Driver.

dbAdminId (String)
The database administrator ID for direct access mode. For example: db2admin.

Note: For a Apache Derby v10.2 embedded database, dbAdminId is not required.

dbAdminPassword (String)
The password associated with the dbAdminId.

Note: For a Apache Derby v10.2 embedded database, dbAdminPassword is not required.

derbySystemHome (String)
The home location of the Apache Derby v10.2 system if you are setting up a Apache Derby v10.2 database.

reportSqlError (String)
Specifies whether to report SQL errors while setting up databases.

file (String)
The full path to a file containing the input parameters. Each input parameter must match a corresponding parameter as it would be typed on the command line, and it must be placed in a `key=value` pair. Each pair must be on a separate line.

dbSchema (String)
The database schema from which you want to delete the federated repository tables. The schema should exist in the database. The default value is the default schema of the database according to the database type. Typically, the default schema is the namespace of the current database user.

The setupIdMgrPropertyExtensionRepositoryTables command:

The setupIdMgrPropertyExtensionRepositoryTables command sets up the property extension repository. The default behavior includes creating and populating the tables in the database.

This command is available in connected or local mode.

Parameters:

schemaLocation (String, Required)
The location of the *app_server_root*`/etc/wim/setup` directory.

laPropXML (String)
The location of the property extension repository definition XML file.

databaseType (String, Required)
The type of database. Supported databases are db2, oracle, informix, derby, sqlserver, db2zos, and db2iseries.

dbURL (String, Required)
The database URL for direct access mode. For example: jdbc:db2:wim.

dbAdminId (String)
The database administrator ID for direct access mode. For example: db2admin.

Note: For a Apache Derby v10.2 embedded database, dbAdminId is not required.

dbAdminPassword (String)
The password associated with the dbAdminId.

Note: For a Apache Derby v10.2 embedded database, dbAdminPassword is not required.

derbySystemHome (String)
The home location of the Apache Derby v10.2 system if you are setting up a Apache Derby v10.2 database.

reportSqlError (String)
Specifies whether to report SQL errors while setting up databases.

skipDBCreation (Boolean)
Specifies whether to create the tables in the property extension repository.

If you set this parameter value to false or do not specify a value, then the command follows the default behavior of creating and populating the tables in the database.

If you set this parameter value to true, manually set up the property extension repository before running this command so that the tables get populated. For more information on this manual process, see the appropriate topic on manually setting up the property extension repository for your database.

file (String)
The full path to a file containing the input parameters. Each input parameter must match a corresponding parameter as it would be typed on the command line, and it must be placed in a `key=value` pair. Each pair must be on a separate line.

dbSchema (String)
The database schema where you want to create the federated repository tables. The schema should exist in the database. The default value is the default schema of the database according to the database type. Typically, the default schema is the namespace of the current database user.

The deleteIdMgrPropertyExtensionRepositoryTables command:

The deleteIdMgrPropertyExtensionRepositoryTables command deletes the tables in the property extension database.

This command is available in the connected or local mode.

Parameters:

schemaLocation (String, Required)
The location of the `<WAS>/etc/wim/setup` directory.

databaseType (String, Required)
The type of database. Supported databases are db2, oracle, informix, derby, sqlserver, db2zos, and db2iseries.

dbURL (String, Required)
The database URL for direct access mode. For example: jdbc:db2:wim.

dbDriver (String)
The name of the database driver. For example: com.ibm.db2.jcc.DB2Driver.

dbAdminId (String)
The database administrator ID for direct access mode. For example: db2admin.

Note: For a Apache Derby v10.2 embedded database, dbAdminId is not required.

dbAdminPassword (String)
The password associated with the dbAdminId.

Note: For a Apache Derby v10.2 embedded database, dbAdminPassword is not required.

derbySystemHome (String)
The home location of the Apache Derby v10.2 system if you are setting up a Apache Derby v10.2 database.

reportSqlError (String)
Specifies whether to report SQL errors while setting up databases.

file (String)
The full path to a file containing the input parameters. Each input parameter must match a corresponding parameter as it would be typed on the command line, and it must be placed in a `key=value` pair. Each pair must be on a separate line.

dbSchema (String)
The database schema from which you want to delete the federated repository tables. The schema should exist in the database. The default value is the default schema of the database according to the database type. Typically, the default schema is the namespace of the current database user.

The setupIdMgrEntryMappingRepositoryTables command:

The setupIdMgrEntryMappingRepositoryTables command sets up the entry mapping repository, which includes creating and populating the tables of the repository.

Parameters:

schemaLocation (String, Required)
The location of the `<WAS>/etc/wim/setup` directory.

databaseType (String, Required)
The type of database. Supported databases are db2, oracle, informix, derby, sqlserver, db2zos, and db2iseries.

dbURL (String, Required)
The database URL for direct access mode. For example: jdbc:db2:wim.

dbDriver (String)
The name of the database driver. For example: com.ibm.db2.jcc.DB2Driver.

dbAdminId (String)
The database administrator ID for direct access mode. For example: db2admin.

Note: For a Apache Derby v10.2 embedded database, dbAdminId is not required.

dbAdminPassword (String)
The password associated with the dbAdminId.

Note: For a Apache Derby v10.2 embedded database, dbAdminPassword is not required.

derbySystemHome (String)
The home location of the Apache Derby v10.2 system if you are setting up a Apache Derby v10.2 database.

reportSqlError (String)
Specifies whether to report SQL errors while setting up databases.

file (String)
The full path to a file containing the input parameters. Each input parameter must match a corresponding parameter as it would be typed on the command line, and it must be placed in a `key=value` pair. Each pair must be on a separate line.

dbSchema (String)
The database schema where you want to create the federated repository tables. The schema

should exist in the database. The default value is the default schema of the database according to the database type. Typically, the default schema is the namespace of the current database user.

The deleteIdMgrEntryMappingRepositoryTables command:

The deleteIdMgrEntryMappingRepositoryTables command deletes the tables in the entry mapping repository.

Parameters:

schemaLocation (String, Required)
The location of the `<WAS>/etc/wim/setup` directory.

databaseType (String, Required)
The type of database. Supported databases are db2, oracle, informix, derby, sqlserver, db2zos, and db2iseries.

dbURL (String, Required)
The database URL for direct access mode. For example: jdbc:db2:wim.

dbDriver (String)
The name of the database driver. For example: com.ibm.db2.jcc.DB2Driver.

dbAdminId (String)
The database administrator ID for direct access mode. For example: db2admin.

Note: For a Apache Derby v10.2 embedded database, dbAdminId is not required.

dbAdminPassword (String)
The password associated with the dbAdminId.

Note: For a Apache Derby v10.2 embedded database, dbAdminPassword is not required.

derbySystemHome (String)
The home location of the Apache Derby v10.2 system if you are setting up a Apache Derby v10.2 database.

reportSqlError (String)
Specifies whether to report SQL errors while setting up databases.

file (String)
The full path to a file containing the input parameters. Each input parameter must match a corresponding parameter as it would be typed on the command line, and it must be placed in a `key=value` pair. Each pair must be on a separate line.

dbSchema (String)
The database schema from which you want to delete the federated repository tables. The schema should exist in the database. The default value is the default schema of the database according to the database type. Typically, the default schema is the namespace of the current database user.

Sample command line usage:

To set up a database using the command line, enter the following:

```
$AdminTask setupIdMgrDBTables {-schemaLocation "C:/WAS/etc/wim/setup" -dbPropXML
"C:/WAS/etc/wim/setup/wimdbproperties.xml" -databaseType db2
-dbURL jdbc:db2:wim -dbAdminId db2admin
-dbDriver com.ibm.db2.jcc.DB2Driver -dbAdminPassword db2adminPwd
-reportSqlError true}
```

To delete database tables using the command line, enter the following:

```
$AdminTask deleteIdMgrDBTables {-schemaLocation "C:/WAS/etc/wim/setup"
-databaseType db2  -dbURL jdbc:db2:wim -dbAdminId db2admin
-dbDriver com.ibm.db2.jcc.DB2Driver -dbAdminPassword db2adminPwd
-reportSqlError true}
```

To set up a property extension repository using the command line, enter the following:

```
$AdminTask setupIdMgrPropertyExtensionRepositoryTables {-schemaLocation
"C:/WAS/etc/wim/setup"
-laPropXML "C:/WAS/etc/wim/setup/wimlaproperties.xml" -databaseType db2
-dbURL jdbc:db2:wim -dbAdminId db2admin -dbDriver com.ibm.db2.jcc.DB2Driver
-dbAdminPassword db2adminPwd -reportSqlError true}
```

To delete a property extension repository using the command line, enter the following:

```
$AdminTask deleteIdMgrPropertyExtensionRepositoryTables {-schemaLocation "C:/WAS/etc/wim/setup "
-databaseType db2 -dbURL jdbc:db2:wim -dbAdminId db2admin -dbDriver
com.ibm.db2.jcc.DB2Driver -dbAdminPassword db2adminPwd -reportSqlError true}
```

To set up an entry mapping repository using the command line, enter the following:

```
$AdminTask setupIdMgrEntryMappingRepositoryTables {-schemaLocation "C:/WAS/etc/wim/setup"
-databaseType db2 -dbURL jdbc:db2:wim -dbAdminId db2admin -dbDriver
com.ibm.db2.jcc.DB2Driver -dbAdminPassword db2adminPwd -reportSqlError true}
```

To delete an entry mapping repository using the command line, enter the following:

```
$AdminTask deleteIdMgrEntryMappingRepositoryTables {-schemaLocation "C:/WAS/etc/wim/setup"
-databaseType db2 -dbURL jdbc:db2:wim -dbAdminId db2admin -dbDriver
com.ibm.db2.jcc.DB2Driver -dbAdminPassword db2adminPwd -reportSqlError true}
```

Sample CLI Usage using -file option:

To set up a database with the `-file` option using the example `params.txt` file below, enter the following:

```
$AdminTask setupIdMgrDBTables {–file C:/params.txt -dbPropXML
"C:/OverrideDBPropParam/wimdbproperties.xml"}
```

Params.txt

```
schemaLocation=C:/WAS/etc/wim/setup
dbPropXML=C:/Program Files/IBM/WebSphere/AppServer/profiles/default
/config/cells/mycell/wim/config/wimdbproperties.xml
laPropXML=C:/Program Files/IBM/WebSphere/AppServer/profiles/default
/config/cells/mycell/wim/config/wimlaproperties.xml
databaseType=db2
dbURL=jdbc:db2:wim
dbDriver=com.ibm.db2.jcc.DB2Driver
reportSqlError=true
dn=o=db.com
dbAdminId=db2admin
dbAdminPassword=dbPassword
wasAdminId=wasadmin
wasAdminPassword=wasadmin1
```

To set up a database with the `-file` option using a file only, enter the following:

```
$AdminTask setupIdMgrDBTables {-file C:/params.txt}
```

Note: The use of a file only works if `-file` is the only parameter specified on the command line. If other parameters are specified then the file is completely ignored, and only the parameters on the command line are used to execute the command.

Manually setting up the property extension repository for federated repositories:

You can use the createIdMgrPropExtDbTables script to create tables in the property extension repository for federated repositories.

Before you begin

The following databases are supported by the script when the database exists on a distributed operating system:

- IBM DB2
- Apache Derby
- IBM Informix Dynamic Server
- Oracle 11g

- Microsoft SQL Server

For a list of the supported database versions, see the IBM WebSphere Application Server detailed system requirements.

To use the IBM DB2 on z/OS or IBM DB2 on iSeries® database, read about manually setting up the property extension repository in DB2.

If you do not have WebSphere Application Server installed on the same system on which you are setting up the database, you must copy the following files from a system where WebSphere Application Server is installed to the system on which you are setting up the database. Ensure that you replicate the same directory structure within the setup directory. The *db_type* variable represents one of the following directory names: `db2`, `oracle`, `informix`, `derby`, or `sqlserver`.

app_server_root\etc\wim\setup\bin\createIdMgrPropExtDbTables.sh
app_server_root\etc\wim\setup\bin\createIdMgrPropExtDbTables.bat
app_server_root\etc\wim\setup\lookaside*db_type*\dbclean.sql
app_server_root\etc\wim\setup\lookaside*db_type*\schema.sql
app_server_root\etc\wim\setup\lookaside*db_type*\primarykeys.sql
app_server_root\etc\wim\setup\lookaside*db_type*\indexes.sql
app_server_root\etc\wim\setup\lookaside*db_type*\references.sql
app_server_root\etc\wim\setup\lookaside\keys.sql
app_server_root\etc\wim\setup\lookaside\bootstrap.sql

Specifying the database schema:

You can specify the database schema where you want to create the federated repository tables when you are manually setting up the property extension repository.

If you want to use the default schema of the database, you must execute the following commands without specifying the DBSCHEMA parameter. Typically, the default schema is the namespace of the current database user.

Complete these steps to replace the schema variable in the SQL files with the actual database schema name. If WebSphere Application Server and the database are not on the same system, set the SCHEMA_LOCATION value to the location where you copied the SQL files.

Windows operating systems:

1. Open a command window.
2. Change to the *app_server_root*\etc\wim\setup directory.
3. Enter the following commands:

```
set SCHEMA_LOCATION=app_server_root\etc\wim\setup\lookaside
set DBTYPE=<db_type>
set DBSCHEMA=dbschemaname
set SCHEMA_DEST_LOCATION=<location where the updated SQL files with replaced variables should be copied>
ws_ant.bat -f app_server_root\etc\wim\setup\filterbuild.xml
```

   where the value of <*db_type*> is db2, derby, informix, oracle, or sqlserver.

Note: : If SCHEMA_DEST_LOCATION is not set, the updated SQL files are copied to a directory with the name as the value not substituted under the current directory. The output shows where the files are copied.

AIX, HP-UX, Linux, and Solaris operating systems:

1. Open a command window
2. Change to the *app_server_root*/etc/wim/setup directory.
3. Enter the following commands:

```
export SCHEMA_LOCATION=app_server_root/etc/wim/setup/lookaside
export DBTYPE=<db_type>
export DBSCHEMA=dbschemaname
export SCHEMA_DEST_LOCATION=<location where the updated SQL files with replaced variables should be copied>
ws_ant.sh -f app_server_root/etc/wim/setup/filterbuild.xml
```

where the value of <*db_type*> is db2, derby, informix, oracle, or sqlserver.

Note: If SCHEMA_DEST_LOCATION is not set, the updated SQL files are copied to a directory with the name as the value not substituted under the current directory. The output shows where the files are copied.

About this task

The following notes apply to specific databases:

- **Oracle 11g**
  - If you did not create the default database when you installed Oracle product, you must manually create the database before you run the createIdMgrPropExtDbTables script. The value of the *ORACLE_SID* variable is the same value as the name of the database.
  - If you want to create the tables in the schema that you specified using DBSCHEMA (described in the previous section, Specifying the database schema) ensure that you create the specified schema in this database before you run the createIdMgrPropExtDbTables script.
  - On the AIX, HP-UX, Linux, and Solaris operating systems, run the createIdMgrPropExtDbTables script either as an Oracle user or as a root user with database administrator (dba) rights and appropriate permissions to run SQL queries as a system database administrator (sysdba).
- **IBM DB2**
  - On the Windows operating systems, you must initialize the DB2 environment before you run the createIdMgrPropExtDbTables script. At the Windows command prompt, enter db2cmd to open a new DB2 command window and run the createIdMgrPropExtDbTables batch file from this prompt.
- **Microsoft SQL Server**
  - Open a command window, change to the *app_server_root*\bin directory, and enter the following commands to replace the variables in the SQL files. If WebSphere Application Server and the database are not on the same system, set the **SCHEMA_LOCATION** value to the location where you copied the SQL files.

    ```
    set SCHEMA_LOCATION=app_server_root\etc\wim\setup\lookaside
    set DBTYPE=sqlserver
    set SCHEMA_DEST_LOCATION=<location where the updated SQL files with replaced variables should be copied>
    set DBOWNER=dbo
    ws_ant.bat -f app_server_root\etc\wim\setup\filterbuild.xml
    ```

    Note: If SCHEMA_DEST_LOCATION is not set, the updated SQL files are copied to a directory with the name as the value not substituted under the current directory. The output shows where the files are copied.

The following default instance is created as a part of the database installation:

- DB2: *DB2*
- Informix: *demo_on*
- SQL Server: *%computername%*

The Informix database is created with the following environment:

```
CLIENT_LOCALE=EN_US.CP1252
DB_LOCALE=EN_US.8859-1
SERVER_LOCALE=EN_US.CP1252
DBLANG=EN_US.CP125
```

Procedure

Run the createIdMgrPropExtDbTables.sh script or createIdMgrPropExtDbTables.bat script to create the tables in the property extension repository.
Run the script from the following location or from the directory to which you previously copied the script file:

AIX, HP-UX, Linux, and Solaris operating systems
app_server_root/etc/wim/setup/bin/createIdMgrPropExtDbTables.sh

Windows
app_server_root\etc\wim\setup\bin\createIdMgrPropExtDbTables.bat

Use the following parameters to specify the values that you require when you run the script:

-b Use this parameter to specify the home directory of the database.

This value is a string value that is required for all database types.

-d Use this parameter to specify the schema of the database.

The value of this parameter should be the same value that you specified for DBSCHEMA (described in the previous section, Specifying the database schema).

This value is a string value that is optional for DB2, Derby, and SQL Server databases, if you want to specify the database schema where you want to create the federated repository tables. This value is not required for Oracle and Informix databases.

-h Use this parameter to display the help information. (Optional)

-i Use this parameter to specify the home directory of the database instance.

This value is a string value that is required for a DB2 database only; do not specify a value for other database types.

This parameter applies to the AIX, HP-UX, Linux, and Solaris operating systems.

-n Use this parameter to specify the name of the database to which you are connecting.

For an Oracle database, the value of the *ORACLE_SID* variable is the same as the name of the database.

This value is a string value that is required for all database types.

-p Use this parameter to specify the password of the database administrator.

This value is a string value that is required for DB2, Oracle, Informix, and SQL Server databases only; do not specify a value for a Derby database.

-s On AIX, HP-UX, Linux, and Solaris operating systems, this parameter specifies the location of the *app_server_root*/etc/wim/setup directory, or the location to which the updated files are copied according to the steps in the previous section, Specifying the database schema.

On Windows operating systems, this parameter specifies the location of the *app_server_root*\etc\wim\setup directory, or the location to which the updated files are copied according to the steps in the previous section, Specifying the database schema.

This value is a string value that is required for all database types.

-t Use this parameter to specify a database type.

- On the AIX, HP-UX, Linux, and Solaris operating systems, specify one of the following valid values: db2, oracle, informix, derby.
- On the Windows operating systems, specify one of the following valid values: db2, oracle, informix, derby, or sqlserver.

This value is a string value that is required for all database types.

-u Use this parameter to specify the user ID of the database administrator.

This value is a string value that is required for DB2, Oracle, Informix, and SQL Server databases only; do not specify a value for a Derby database.

Example

Run the appropriate script for your database and operating system to create tables in the property extension repository. Use the sample values to specify database parameters. If the database exists on a system where WebSphere Application Server is not installed, the following examples assume that your PATH variable includes an entry for the location to which you copied the script files. For the AIX, HP-UX, Linux, and Solaris operating systems, the entry might be the *app_server_root*/etc/wim/setup/bin/ or the /setup/bin/ directory. For Windows operating systems, the entry might be the *app_server_root*\etc\wim\setup\bin\ or the \setup\bin\ directory.

The examples in the following section are organized into multiple lines for illustration purposes only.

On the AIX, HP-UX, Linux, and Solaris operating systems:

Oracle databases

```
createIdMgrPropExtDbTables.sh
-b /space/oracle/product/10.2.0/Db_1/
-n orcl
-u system
-p manager
-s /opt/IBM/WebSphere/AppServer1/etc/wim/setup
-t oracle
```

Informix databases

```
createIdMgrPropExtDbTables.sh
-b /opt/IBM/informix/
-n demo_on
-u informix
-p informix
-s /opt/IBM/WebSphere/AppServer/etc/wim/setup
-t informix
```

DB2 databases

```
createIdMgrPropExtDbTables.sh
-b /opt/ibm/db2/V9.1/
-n db2inst1
-p db2inst1
-s /opt/IBM/WebSphere/AppServer/etc/wim/setup
-t DB2
-u db2inst1
-i /home/db2inst1/
```

Derby databases

```
createIdMgrPropExtDbTables.sh
-b /opt/ibm/derby/
-n test11
-s /opt/IBM/WebSphere/AppServer/etc/wim/setup
-t derby
```

On the Windows operating systems:

Oracle databases

```
createIdMgrPropExtDbTables.bat
-b "c:\oracle\product\10.2.0\Db_1"
-n orcl
-u system
-p manager
-s "c:\Program Files\IBM\WebSphere\AppServer1\etc\wim\setup"
-t oracle
```

Informix databases

```
createIdMgrPropExtDbTables.bat
-b "c:\Program Files\IBM\informix"
-n demo_on
-u informix
-p informix
-s "c:\Program Files\IBM\WebSphere\AppServer\etc\wim\setup"
-t informix
```

DB2 databases

```
createIdMgrPropExtDbTables.bat
-t db2
-u db2admin
-p sec001ret#
-n test23
-b "c:\Program Files\IBM\SQLLIB"
-s "c:\Program Files\IBM\WebSphere\AppServer1\etc\wim\setup"
```

Derby databases

```
createIdMgrPropExtDbTables.bat
-t derby
-b "c:\Derby"
-n test11
-s "c:\Program Files\IBM\WebSphere\AppServer1\etc\wim\setup"
```

Microsoft SQL Server databases

```
createIdMgrPropExtDbTables.bat
-t sqlserver
-u sa
-p sec001ret#
-n sqlsrv
-b "c:\Progra~1\Micros~1\90\Tools"
-s "C:\Progra~1\IBM\WebSphere\AppServer1\etc\wim\setup"
```

What to do next

Run the **setupIdMgrPropertyExtensionRepositoryTables** command with the **skipDBCreation** parameter set to true to populate the tables that are created. For more information, read about setting up an entry mapping repository, a property extension repository, or a custom registry database repository using wsadmin commands.

Manually setting up the property extension repository for DB2 for iSeries or DB2 for z/OS:

Use this task to set up the property extension repository for DB2 for iSeries or DB2 for z/OS.

Before you begin

The information in this topic applies in the following scenarios:

- The application server and the database both exist on the IBM i operating system.
- The application server and the database both exist on the z/OS operating system.
- The application server exists on a distributed operating system, but the database exists on either the IBM i or z/OS operating system.

If you do not have WebSphere Application Server installed in the system on which you are setting up the database, copy the following files from a system where WebSphere Application Server is installed to the system on which you are setting up the database:

DB2 for iSeries

app_server_root/etc/wim/setup/lookaside/db2iseries/dbclean.sql
app_server_root/etc/wim/setup/lookaside/db2iseries/schema.sql
app_server_root/etc/wim/setup/lookaside/db2iseries/primarykeys.sql
app_server_root/etc/wim/setup/lookaside/db2iseries/indexes.sql

app_server_root/etc/wim/setup/lookaside/db2iseries/references.sql
app_server_root/etc/wim/setup/lookaside/keys.sql
app_server_root/etc/wim/setup/lookaside/bootstrap.sql

DB2 for z/OS

app_server_root/etc/wim/setup/lookaside/db2zos/dbclean.sql
app_server_root/etc/wim/setup/lookaside/db2zos/schema.sql
app_server_root/etc/wim/setup/lookaside/db2zos/primarykeys.sql
app_server_root/etc/wim/setup/lookaside/db2zos/indexes.sql
app_server_root/etc/wim/setup/lookaside/db2zos/references.sql
app_server_root/etc/wim/setup/lookaside/keys.sql
app_server_root/etc/wim/setup/lookaside/bootstrap.sql

About this task

For information about how to create a database and run SQL queries in DB2 for iSeries, see the DB2 Universal Database™ for iSeries in the IBM iSeries Information Center.

For information about how to create a database and run SQL queries in DB2 for z/OS, see the Information Management Software for z/OS Solutions Information Center.

Procedure

1. Open a command window.
2. Change to the *app_server_root*/bin directory
3. Enter the following commands to replace the variables in the SQL files:
   a. `export SCHEMA_LOCATION=`*app_server_root*`/etc/wim/setup/lookaside`

      Set the SCHEMA_LOCATION value to the location where you copied the SQL files if you do not have WebSphere Application Server installed on the same system on which you are setting up the database.

   b. `export DBTYPE=<`*db_type*`>`

      where the value of *<db_type>* is db2iseries or db2zos

   c. To specify the database schema where you want to create the federated repository tables use the DBSCHEMA command. If you want to use the default schema, which is typically the namespace of the current database user, do not specify the DBSCHEMA command.

      `export DBSCHEMA=dbschemaname`

   d. `export TSPREFIX=<`*tsprefix*`>`

      where *<tsprefix>* is the tablespace prefix. The maximum length allowed for this string is 3 characters.

   e. `export SCHEMA_DEST_LOCATION=<`*schema_dest_location*`>`

      where *<schema_dest_location>* is the location where the updated SQL files with replaced variables should be copied. If SCHEMA_DEST_LOCATION is not set, the updated SQL files are copied to a directory with the name as the unsubstituted value under the current directory. The output indicates where the files are copied to.

   f. `./ws_ant.sh -f `*app_server_root*`/etc/wim/setup/filterbuild.xml`
4. Start the DB2 server.
5. Create a database.
6. Run the SQL files, which were previously referenced, to create the tables for the property extension repository. If you are setting up the database on the same system on which the application server is installed, the files are located in the following locations:

   DB2 for iSeries

   app_server_root/etc/wim/setup/lookaside/db2iseries/dbclean.sql
   app_server_root/etc/wim/setup/lookaside/db2iseries/schema.sql
   app_server_root/etc/wim/setup/lookaside/db2iseries/primarykeys.sql
   app_server_root/etc/wim/setup/lookaside/db2iseries/indexes.sql

app_server_root/etc/wim/setup/lookaside/db2iseries/references.sql
app_server_root/etc/wim/setup/lookaside/keys.sql
app_server_root/etc/wim/setup/lookaside/bootstrap.sql

DB2 for z/OS

app_server_root/etc/wim/setup/lookaside/db2zos/dbclean.sql
app_server_root/etc/wim/setup/lookaside/db2zos/schema.sql
app_server_root/etc/wim/setup/lookaside/db2zos/primarykeys.sql
app_server_root/etc/wim/setup/lookaside/db2zos/indexes.sql
app_server_root/etc/wim/setup/lookaside/db2zos/references.sql
app_server_root/etc/wim/setup/lookaside/keys.sql
app_server_root/etc/wim/setup/lookaside/bootstrap.sql

Otherwise, run the SQL files from the location to which you copied the files. If you executed the commands to substitute variables according to the steps in the previous section, Specifying the database schema, the SQL files are copied to the location you specified for SCHEMA_DEST_LOCATION. If SCHEMA_DEST_LOCATION is not set, the updated SQL files are copied to a directory with the name as the unsubstituted value under the current directory. The output shows where the files are copied.

What to do next

Run the **setupIdMgrPropertyExtensionRepositoryTables** command with the **skipDBCreation** parameter set to true to populate the tables that are created. For more information, read about setting up an entry mapping repository, a property extension repository, or a custom registry database repository using wsadmin commands.

Configuring the WebSphere Application Server data source:

Installed applications use data sources as resources to obtain connection to relational databases. To create these connections between an application and a relational database, WebSphere Application Server uses the driver implementation classes that are encapsulated by the JDBC provider, which is an object that represents vendor-specific JDBC driver classes to WebSphere Application Server. For access to a relational databases, applications use the JDBC drivers and data sources that you configure for WebSphere Application Server.

Procedure

1. Start the WebSphere Application Server administrative console.
2. Click **Security -> Global security**.
3. On the Configuration panel, under Authentication, expand Java Authentication and Authorization Service and click **J2C authentication data**.
4. Click **New** and enter the Alias, User ID and Password.
5. Click **Ok**.
6. On the WebSphere Application Server administrative console, expand **Resources**. Expand **JDBC** then click **JDBC Providers**.
7. In the Scope section, choose the **Node level** from the drop-down list.
8. Click **New** to create a new JDBC driver.
9. Select, in this order, the Database type, Provider type, Implementation type and Name. The Name automatically fills based on the implementation type you choose.
10. Click **Next** and configure the database class path. Click **Next**.
11. On the Summary page, click **Finish**.
12. Click **Save** to save your selections. The JDBC providers page then appears.
13. On the WebSphere Application Server administrative console, click **Data sources**.

14. Click **New** to create a new data source. Enter the Data source name and the JNDI name, and choose the authentication alias from the drop-down list in Component-managed authentication alias. The JNDI name should match the datasourceName value set in `wimconfig.xml`. By default, it is `jdbc/wimDS`.

    Note: For Apache Derby v10.2 embedded databases, leave the Component-managed authentication alias field set to `NONE`.
15. Click **Next**.
16. Enter the Database name and deselect the checkbox, **Use this data source in container managed persistence (CMP)**. Click **Next**.
17. On the Summary page, click **Finish**.
18. The Data sources page displays. Click **Save**, Then select the check box for the authentication alias previously created. Click **Test Connection**. The message should indicate that the connection is successful. Ignore any warnings, and then click **Next**.
19. Save the configurations, and restart WebSphere Application Server.

Configuring an entry mapping repository in a federated repository configuration

Follow this task to configure an entry mapping repository that is used to store data for managing profiles on multiple repositories.

About this task

An *entry-level join* means that the federated repository configuration uses multiple repositories simultaneously and recognizes the entries in the different repositories as entries representing distinct entities. For example, a company might have a Lightweight Directory Access Protocol (LDAP) directory that contains entries for its employees and a database that contains entries for business partners and customers. By configuring an entry mapping repository, a federated repository configuration can use both the LDAP and the database at the same time. The federated repository configuration hierarchy and constraints for identifiers provide the aggregated namespace for both of those repositories and prevent identifiers from colliding.

When you configure an entry mapping repository, you can supply a valid data source, a direct connection configuration, or both. The system first tries to connect by way of the data source. If the data source is not available, then the system uses the direct access configuration.

Restriction: You cannot configure an entry mapping repository in a mixed-version deployment manager cell.

Procedure

1. Configure the WebSphere Application Server data source. See “Configuring the WebSphere Application Server data source” on page 291.
2. Set up the entry mapping repository using wsadmin. See “Setting up an entry mapping repository, a property extension repository, or a custom registry database repository using wsadmin commands” on page 277; ignore the “Before you begin” options.
3. Configure the entry mapping repository into the federated repository by doing the following:
   a. In the administrative console, click **Security** > **Global security**.
   b. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**. To configure for a specific domain in a multiple security domain environment, click **Security domains** > **domain_name**. Under Security Attributes, expand **User Realm**, and click **Customize for this domain**. Select the Realm type as **Federated repositories** and then click **Configure**.
   c. Click **Entry mapping repository**.
   d. Supply the name of the data source in the Data source name field.

e. Select the type of database that is used for the property extension repository.

f. Supply the name of the Java database connectivity (JDBC) driver in the JDBC driver field.

Values include:

DB2 `com.ibm.db2.jcc.DB2Driver`

Informix
`com.informix.jdbc.IfxDriver`

DataDirect Connect
com.ddtek.jdbc.sqlserver.SQLServerDriver

Derby `org.apache.derby.jdbc.EmbeddedDriver`

Microsoft SQL Server
`com.microsoft.sqlserver.jdbc.SQLServerDriver`

Oracle
`oracle.jdbc.driver.OracleDriver`

g. Supply the database URL that is used to access the property extension repository with JDBC in the Database URL field. Use an alphanumeric text string that conforms to the standard JDBC URL syntax.

Values include:

DB2 `jdbc:db2:wim`

Informix
`jdbc:informix-sqli://`*`host_name`*`:1526/wim:INFORMIXSERVER=`*`IFXServerName`*`;`

DataDirect Connect
`jdbc:datadirect:sqlserver://`*`host_name`*`:1433;databaseName=wim;selectMethod=cursor;`

Derby `jdbc:derby:c:\derby\wim`

Microsoft SQL Server
`jdbc:sqlserver://`*`host_name`*`:1433;databaseName=wim;selectMethod=cursor;`

Oracle
`jdbc:oracle:thin:@`*`host_name`*`:`*`port`*`:`*`dbname`*

h. Supply the user name of the database administrator in the Database administrator user name field.

i. Supply the password of the database administrator in the Password field.

j. Click **OK**.

Results

After completing these steps, your federated repository configuration, which includes an entry mapping repository, is configured.

What to do next

1. After configuring the federated repositories, click **Security > Global security** to return to the Global security panel. Verify that Federated repositories is identified in the Current realm definition field. If Federated repositories is not identified, select **Federated repositories** from the Available realm definitions field and click **Set as current**. To verify the federated repositories configuration, click **Apply** on the Global security panel. If Federated repositories is not identified in the Current realm definition field, your federated repositories configuration is not used by WebSphere Application Server.
2. If you are enabling security, complete the remaining steps as specified in “Enabling security for the realm” on page 84. As the final step, validate this setup by clicking **Apply** in the Global security panel.
3. Save, stop, and restart all the product servers (deployment managers, nodes, and Application Servers) for changes in this panel to take effect. If the server comes up without any problems, the setup is correct.

Entry mapping repository settings:

Use this page to configure an entry mapping repository that is used to store data for managing profiles on multiple repositories.

To view this administrative console page, complete the following steps:

1. In the administrative console, click **Security > Global security**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**. To configure for a specific domain in a multiple security domain environment, click **Security domains > *domain_name***. Under Security Attributes, expand User Realm, and click **Customize for this domain**. Select the Realm type as **Federated repositories** and then click **Configure**.
3. Under Additional properties, click **Entry mapping repository**.

When you finish adding or updating your federated repository configuration, go to the **Security > Global security** panel and click **Apply** to validate the changes.

Data source name:

Specifies the Java Naming and Directory Interface (JNDI) name of the data source that is used to access the entry mapping repository.

Information	**Value**
Default:	`jdbc/wimDS`

Database type:

Specifies the type of database that is used to access the entry mapping repository.

Information	**Value**
Default:	`DB2`

JDBC driver:

Specifies the Java Database Connectivity (JDBC) driver that is used to access the entry mapping repository.

Values include:

DB2 `com.ibm.db2.jcc.DB2Driver`

DataDirect Connect
`com.ddtek.jdbc.sqlserver.SQLServerDriver`

Informix
`com.informix.jdbc.IfxDriver`

Oracle
`oracle.jdbc.driver.OracleDriver`

Microsoft SQL Server
`com.microsoft.sqlserver.jdbc.SQLServerDriver`

Derby `org.apache.derby.jdbc.EmbeddedDriver`

Database URL:

Specifies the web address for the entry mapping repository.

Values include:

DB2 `jdbc:db2:wim`

Derby `jdbc:derby:c:\derby\wim`

DataDirect Connect
`datadirect:sqlserver://:host_name1433;databaseName=wim;selectMethod=cursor;`

Oracle
`jdbc:oracle:thin:@host_name:port:dbname`

Microsoft SQL Server
`jdbc:sqlserver://host_name:1433;databaseName=wim;selectMethod=cursor;`

Informix
`jdbc:informix-sqli://host_name:port/wim:INFORMIXSERVER=IFXServerName;`

Database administrator user name:

Specifies the user name of the database administrator that is used to access the entry mapping repository.

Password:

Specifies the password that is used to enable the database administrator to access the entry mapping repository.

Configuring supported entity types in a federated repository configuration

Follow this task to configure supported entity types for user and group management.

About this task

You must configure the supported entity types before you can manage this account with Users and Groups in the administrative console. The supported entity types are Group, OrgContainer, and PersonAccount. A Group entity represents a simple collection of entities that might not have any relational context. An OrgContainer entity represents an organization, such as a company or an enterprise, a subsidiary, or an organizational unit, such as a division, a location, or a department. A PersonAccount entity represents a human being. You cannot add or delete the supported entity types, because these types are predefined.

The Base entry for the default parent determines the repository location where entities of the specified type are placed on write operations by user and group management.

Note: To manage users and groups, click **Users and Groups** in the console navigation tree. Click either **Manage Users** or **Manage Groups**. To manage users and groups for a specific domain in a multiple security domain environment, click **Security > Global security > Security Domains > *domain_name***. Under Security Attributes, expand User Realm, and click **Customize for this domain**. Select the Realm type as **Federated repositories**. Click **Apply** and **Save** to the master configuration. On Security domains panel that appears, click the *domain_name* again to go to the domain configuration panel. Under User realm, click the **Manage users** or **Manager Groups** links that are displayed now. These links to manage users and groups for a specific domain are displayed only after you save the federated repositories configuration for the domain.

Note: You must restart the server or dmgr if the federated repository has changed before using the Manage Users option. Otherwise, user or group changes made to the repository could be lost after restart.

Procedure

1. In the administrative console, click **Security > Global security**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**. To configure for a specific domain in a multiple security domain environment, click **Security domains > *domain_name***. Under Security Attributes, expand User Realm, and click **Customize for this domain**. Select the Realm type as **Federated repositories** and then click **Configure**.
3. Click **Supported entity types** to view a list of predefined entity types.
4. Click the name of a predefined entity type to change its configuration.
5. Supply the distinguished name of a base entry in the repository in the Base entry for the default parent field. This entry determines the default location in the repository where entities of this type are placed on write operations by user and group management.
6. Supply the relative distinguished name (RDN®) properties for the specified entity type in the Relative Distinguished Name properties field. Possible values are `cn` for Group, `uid` or `cn` for PersonAccount, and `o`, `ou`, `dc`, and `cn` for OrgContainer. Delimit multiple properties for the OrgContainer entity with a semicolon (;).

   The following list outlines known requirements and limitations that apply to specific Lightweight Directory Access Protocol (LDAP) servers:

   Using Microsoft Active Directory as the LDAP server

   - Unless you modify the LDAP schema to use `uid`, you must specify `cn` in the Relative Distinguished Name (RDN) properties field for the PersonAccount entity type.
   - Secure Sockets Layer communications must be enabled to create users with passwords. To select the **Require SSL communications** option, see the topic “Configuring Lightweight Directory Access Protocol in a federated repository configuration” on page 256.
   - Typically the value of `user` is specified as the value in the Object classes field for the PersonAccount entity type and the value of `group` is specified as the value in the Object classes field for the Group entity type.

   Using a Lotus Domino Enterprise Server as the LDAP server

   - Typically, the value of `cn` is specified in the Relative Distinguished Name (RDN) properties field for the PersonAccount entity type. The value of `uid` is also acceptable.
   - Typically, both `inetOrgPerson` and `dominoPerson` are used as values in the Object classes field for the PersonAccount entity type.

   Using Sun ONE Directory Server as the LDAP server

   - Typically, `groupOfUniqueNames` is specified as the value in the Object classes field for the Group entity type.
7. Click **OK**.

Results

After completing these steps, your federated repository configuration, which uses supported entity types, is configured.

What to do next

1. After configuring the federated repositories, click **Security > Global security** to return to the Global security panel. Verify that Federated repositories is identified in the Current realm definition field. If Federated repositories is not identified, select **Federated repositories** from the Available realm definitions field and click **Set as current**. To verify the federated repositories configuration, click **Apply** on the Global security panel. If Federated repositories is not identified in the Current realm definition field, your federated repositories configuration is not used by WebSphere Application Server.
2. If you are enabling security, complete the remaining steps as specified in “Enabling security for the realm” on page 84. As the final step, validate this setup by clicking **Apply** on the Global security panel.

3. Save, stop, and restart all the product servers (deployment managers, nodes, and Application Servers) for changes in this panel to take effect. If the server comes up without any problems, the setup is correct.

Supported entity types collection:

Use this page to list entity types that are supported by the member repositories or to select an entity type to view or change its configuration properties.

To view this administrative console page, complete the following steps:

1. In the administrative console, click **Security > Global security**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**. To configure for a specific domain in a multiple security domain environment, click **Security domains > *domain_name***. Under Security Attributes, expand User Realm, and click **Customize for this domain**. Select the Realm type as **Federated repositories** and then click **Configure**.
3. Under Additional properties, click **Supported entity types**.

You must configure the supported entity types before you can manage this account with Users and Groups in the administrative console. The Base entry for the default parent determines the repository location where entities of the specified type are placed on write operations by user and group management.

When you finish adding or updating your federated repository configuration, go to the **Security > Global security** panel and click **Apply** to validate the changes.

Entity type:

Specifies the entity type name.

Base entry for the default parent:

Specifies the distinguished name of a base entry in the repository.

This entry determines the default location in the repository where entities of this type are placed on write operations by user and group management.

Relative Distinguished Name properties:

Specifies the relative distinguished name (RDN) properties for the specified entity type.

Possible values are `cn` for Group, `uid` or `cn` for PersonAccount, and `o`, `ou`, `dc`, and `cn` for OrgContainer. Delimit multiple properties for the OrgContainer entity with a semicolon (;).

Supported entity types settings:

Use this page to configure entity types that are supported by the member repositories.

To view this administrative console page, complete the following steps:

1. In the administrative console, click **Security > Global security**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**. To configure for a specific domain in a multiple security domain environment, click **Security domains > *domain_name***. Under Security Attributes, expand User Realm, and click **Customize for this domain**. Select the Realm type as **Federated repositories** and then click **Configure**.
3. Under Additional properties, click **Supported entity types**.

4. Click the name of a configured entity type to view or change its configuration.

You must configure the supported entity types before you can manage this account with Users and Groups in the administrative console. The Base entry for the default parent determines the repository location where entities of the specified type are placed on write operations by user and group management.

When you finish adding or updating your federated repository configuration, go to the **Security > Global security** panel and click **Apply** to validate the changes.

Entity type:

Specifies the name of the entity type.

Base entry for the default parent:

Specifies the distinguished name of a base entry in the repository.

This entry determines the default location in the repository where entities of this type are placed on write operations by user and group management.

Relative Distinguished Name properties:

Specifies the relative distinguished name (RDN) properties for the specified entity type.

Possible values are `cn` for Group, `uid` or `cn` for PersonAccount, and `o`, `ou`, `dc`, and `cn` for OrgContainer. Delimit multiple properties for the OrgContainer entity with a semicolon (;).

Configuring user repository attribute mapping in a federated repository configuration

Follow this task to set or modify the mapping for user or group attributes of a user registry to federated repository properties in the current realm.

Procedure

1. In the administrative console, click **Security > Global security**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**. To configure for a specific domain in a multiple security domain environment, click **Security domains > domain_name**. Under Security Attributes, expand **User Realm**, and click **Customize for this domain**. Select the Realm type as **Federated repositories** and then click **Configure**.
3. Under Additional properties, click the **User repository attribute mapping** link.
4. Select an attribute and click **Edit** to modify the mapping.

   Property for Input
   Specifies the name of the federated repository property that maps to the specified user registry attribute when it is an input parameter for the user registry interface.

   Property for Output
   Specifies the name of the federated repository property that maps to the specified user registry attribute when it is an output parameter (return value) for the user registry interface. In most cases, the propertyForInput and propertyForInput would be the same.
5. Click **OK** and **Save** to the master configuration.
6. Restart the application server.

Results

After completing these steps, user or group attributes of the user registry are mapped to federated repository properties in the current realm.

Supported entity types collection:

Use this page to list entity types that are supported by the member repositories or to select an entity type to view or change its configuration properties.

To view this administrative console page, complete the following steps:

1. In the administrative console, click **Security > Global security**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**. To configure for a specific domain in a multiple security domain environment, click **Security domains > *domain_name***. Under Security Attributes, expand User Realm, and click **Customize for this domain**. Select the Realm type as **Federated repositories** and then click **Configure**.
3. Under Additional properties, click **Supported entity types**.

You must configure the supported entity types before you can manage this account with Users and Groups in the administrative console. The Base entry for the default parent determines the repository location where entities of the specified type are placed on write operations by user and group management.

When you finish adding or updating your federated repository configuration, go to the **Security > Global security** panel and click **Apply** to validate the changes.

Entity type:

Specifies the entity type name.

Base entry for the default parent:

Specifies the distinguished name of a base entry in the repository.

This entry determines the default location in the repository where entities of this type are placed on write operations by user and group management.

Relative Distinguished Name properties:

Specifies the relative distinguished name (RDN) properties for the specified entity type.

Possible values are `cn` for Group, `uid` or `cn` for PersonAccount, and `o`, `ou`, `dc`, and `cn` for OrgContainer. Delimit multiple properties for the OrgContainer entity with a semicolon (;).

Supported entity types settings:

Use this page to configure entity types that are supported by the member repositories.

To view this administrative console page, complete the following steps:

1. In the administrative console, click **Security > Global security**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**. To configure for a specific domain in a multiple security domain environment, click **Security domains > *domain_name***. Under Security Attributes, expand User Realm, and click **Customize for this domain**. Select the Realm type as **Federated repositories** and then click **Configure**.

3. Under Additional properties, click **Supported entity types**.
4. Click the name of a configured entity type to view or change its configuration.

You must configure the supported entity types before you can manage this account with Users and Groups in the administrative console. The Base entry for the default parent determines the repository location where entities of the specified type are placed on write operations by user and group management.

When you finish adding or updating your federated repository configuration, go to the **Security > Global security** panel and click **Apply** to validate the changes.

Entity type:

Specifies the name of the entity type.

Base entry for the default parent:

Specifies the distinguished name of a base entry in the repository.

This entry determines the default location in the repository where entities of this type are placed on write operations by user and group management.

Relative Distinguished Name properties:

Specifies the relative distinguished name (RDN) properties for the specified entity type.

Possible values are `cn` for Group, `uid` or `cn` for PersonAccount, and `o`, `ou`, `dc`, and `cn` for OrgContainer. Delimit multiple properties for the OrgContainer entity with a semicolon (;).

Managing repositories in a federated repository configuration

Follow this topic to manage repositories in a federated repository configuration.

Procedure

1. In the administrative console, click **Security > Global security**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**. To configure for a specific domain in a multiple security domain environment, click **Security domains > *domain_name***. Under Security Attributes, expand User Realm, and click **Customize for this domain**. Select the Realm type as **Federated repositories** and then click **Configure**.
3. Under Related items, click **Manage repositories**. Repositories that are configured in the system are listed in the collection panel. This list includes repositories that are configured using the federated repository functionality as well as repositories that are created using wsadmin commands described in the topic IdMgrRepositoryConfig command group for the AdminTask object.
4. Optional: Click **Add** to configure a new external repository and select the repository type as **LDAP repository**, **Custom repository**, or **File repository**.

   Restriction: You cannot add a database repository using the administrative console. This repository configuration is supported by using wsadmin commands only.
5. Optional: Click **Delete** to delete a repository that you specified previously using the administrative console or wsadmin commands.

   Restriction: You cannot delete the built-in, file-based repository from the collection panel.
6. Optional: Select one of the repository identifier entries to view or update an external repository that is configured in the system previously.

- The LDAP repository configuration settings are described in detail in “Configuring Lightweight Directory Access Protocol in a federated repository configuration” on page 256.
- The custom repository configuration settings are described in “Adding a custom repository to a federated repositories configuration” on page 311.
- The file-based repository configuration settings are described in “Adding a file-based repository to a federated repositories configuration” on page 238.

Restriction: While database repositories that are configured in the system are listed in the collection panel, you cannot update a database repository using the administrative console. Updates to a database repository are supported by using wsadmin commands only.

7. Click **OK**.

Results

After completing these steps, the collection panel under Managing repositories reflects a current list of repositories that are configured in your system.

What to do next

1. To add one or more external repositories that are listed on this collection panel into the realm, see “Managing the realm in a federated repository configuration” on page 226.
2. After configuring the federated repositories, click **Security > Global security** to return to the Global security panel. Verify that Federated repositories is identified in the Current realm definition field. If Federated repositories is not identified, select **Federated repositories** from the Available realm definitions field and click **Set as current**. To verify the federated repositories configuration, click **Apply** on the Global security panel. If Federated repositories is not identified in the Current realm definition field, your federated repositories configuration is not used by WebSphere Application Server.
3. If you are enabling security, complete the remaining steps as specified in “Enabling security for the realm” on page 84. As the final step, validate this setup by clicking **Apply** in the Global security panel.
4. Save, stop, and restart all the product servers (deployment managers, nodes, and Application Servers) for changes in this panel to take effect. If the server comes up without any problems, the setup is correct.

Important: Be aware of any changes made to built-in, file-based repositories that are part of the federated repositories configuration. You must replicate these changes to the managed nodes to ensure that your network deployment configuration is synchronized. See “Replicating changes to a built-in, file-based repository” for the steps needed to perform this replication.

Replicating changes to a built-in, file-based repository:

Changes to built-in, file-based repositories are not automatically replicated to managed nodes in a federated repositories configuration. You need to use the administrative console to replicate the changes you make to a built-in, file-based repository.

About this task

The network deployment support in a federated repositories configuration only updates the in-memory state of the processes that are running on the managed nodes. Because WebSphere Application Server synchronizes the file systems, the network deployment support does not attempt to update the file systems of the managed nodes.

You must synchronize the node configuration to replicate the changes to the built-in, file-based repository.

Procedure

1. In the administrative console, click **System Administration > Nodes**. to access the nodes panel.

2. On the Nodes panel, select all the relevant nodes for which the changes to the built-in, file-based repository need to be made.
3. Click **Full Resynchronize**. The resynchronize operation resolves conflicts among configuration files and can take several minutes to complete.

Results

After completing these steps, your federated repository configuration of managed nodes reflects the changes to the built-in, file-based repository.

Manage repositories collection:

Use this page to list repositories that are configured in the system or to select a repository to view or change its configuration properties. You can add or delete external repositories.

To view this administrative console page, complete the following steps:

1. In the administrative console, click **Security > Global security**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**. To configure for a specific domain in a multiple security domain environment, click **Security domains > *domain_name***. Under Security Attributes, expand User Realm, and click **Customize for this domain**. Select the Realm type as **Federated repositories** and then click **Configure**.
3. Under Related items, click **Manage repositories**.

When you finish adding or updating your federated repository configuration, go to the **Security > Global security** panel and click **Apply** to validate the changes.

Repository identifier:

Specifies a unique identifier for the repository. This identifier uniquely identifies the repository within the cell.

Repository type:

Specifies the repository type, such as File or LDAP.

Add:

Select to add a new LDAP, custom or file repository.

Repository reference settings:

Use this page to configure a repository reference. A repository reference is a single repository that contains a set of identity entries that are referenced by a base entry into the directory information tree.

To view this administrative console page, complete the following steps:

1. In the administrative console, click **Security > Global security**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**. To configure for a specific domain in a multiple security domain environment, click **Security domains > *domain_name***. Under Security Attributes, expand User Realm, and click **Customize for this domain**. Select the Realm type as **Federated repositories** and then click **Configure**.
3. Click **Add base entry to realm**.

When you finish adding or updating your federated repository configuration, go to the **Security > Global security** panel and click **Apply** to validate the changes.

Repository:

Specifies a unique identifier for the repository. This identifier uniquely identifies the repository within the cell.

Expand the drop-down list to display a list of previously defined repository identifiers.

Distinguished name of a base entry that uniquely identifies this set of entries in the realm:

Specifies the distinguished name (DN) that uniquely identifies this set of entries in the realm.

If multiple repositories are included in the realm, it is necessary to define an additional distinguished name that uniquely identifies this set of entries within the realm. Overlapping base entries are not supported. You should not define two base entries where one is c=us, and the other is o=myorg,c=us in the same realm; otherwise a search returns duplicate results.

Distinguished name of a base entry in this repository:

Specifies the Lightweight Directory Access Protocol (LDAP) distinguished name (DN) of the base entry within the repository. The entry and its descendents are mapped to the subtree that is identified by the unique base name entry field.

If this field is left blank, then the subtree defaults to the root of the LDAP repository.

Increasing the performance of an LDAP repository in a federated repository configuration

Follow the steps given here to increase the performance of an LDAP repository in a federated repository configuration.

Before you begin

The settings that are available on the Performance panel are independent options that pertain specifically to an LDAP repository configured using the federated repositories functionality. These options do not affect your entire WebSphere Application Server configuration.

Procedure

1. In the administrative console, click **Security > Global security**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**. To configure for a specific domain in a multiple security domain environment, click **Security domains > *domain_name***. Under Security Attributes, expand User Realm, and click **Customize for this domain** . Select the Realm type as **Federated repositories** and then click **Configure**.
3. Under Related items, click **Manage repositories** > *repository_name*.
4. Under Additional properties, click **Performance**.
5. Optional: Select the **Limit search time** option and enter the maximum number of milliseconds that the Application Server can use to search through your Lightweight Directory Access Protocol (LDAP) entries.
6. Optional: Select the **Limit search returns** option and enter the maximum number of entries to return that match the search criteria.
7. Optional: Select the **Use connection pooling** option to specify whether the Application Server can store separate connections to the LDAP server for reuse.

8. Optional: Select the **Enable context pool** option to specify whether multiple applications can use the same connection to the LDAP server. If you select the option, specify the initial, preferred, and maximum number of entries that can use the same connection. The **Enable context pool** option can be enabled either in conjunction with the **Use connection pool** option or separately. If this option is disabled, a new connection is created for each context. You can also select the **Context pool times out** option and specify the number of seconds after which the entries in the context pool expire.
9. Optional: Set the **Maximum size** value of the context pool to zero (0).
10. Optional: Select the **Cache the attributes** option and specify the maximum number of search attribute entries. This option enables WebSphere Application Server to save the LDAP entries so that it can search the entries locally rather than making multiple calls to the LDAP server. Click the **Cache times out** option that is associated with the **Cache the attributes** option to specify the maximum number of seconds that the Application Server can save these entries. Specify the **Distribution policy** for the dynamic attribute cache in a clustered environment as **Not shared**, shared **Push**, or shared **Push and pull**. This setting is read during the adapter startup process and the cache policy is set accordingly.
11. Optional: Select the **Cache the search results** option and specify the maximum number of search result entries. This option enables WebSphere Application Server to save the results of a search inquiry instead of making multiple calls to the LDAP server to search and retrieve the results of that search. Click the **Cache times out** option that is associated with the **Cache the search results** option to specify the maximum number of seconds that the Application Server can save the results. Specify the **Distribution policy** for the dynamic attribute cache in a clustered environment as **Not shared**, shared **Push**, or shared **Push and pull**. This setting is read during the adapter startup process and the cache policy is set accordingly.
12. Optional: Create the root DataObject object locally using the com.ibm.websphere.wim.util.SDOHelper.createRootDataObject method instead of the com.ibm.websphere.wim.ServiceProvider.createRootDataObject method.

Results

These options are available to potentially increase the performance of your federated repositories configuration. However, the any increase in performance is dependant upon your specific configuration.

Lightweight Directory Access Protocol performance settings:

Use this page to minimize impacts to performance by adding opened connections and contexts to internally maintained pools and reusing them. Also minimize performance impacts by maintaining internal caches of retrieved data.

To view this administrative console page, complete the following steps:

1. In the administrative console, click **Security > Global security**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**. To configure for a specific domain in a multiple security domain environment, click **Security domains > *domain_name***. Under Security Attributes, expand User Realm, and click **Customize for this domain**. Select the Realm type as **Federated repositories** and then click **Configure**.
3. Under Related items, click **Manage repositories**.
4. Click **Add** to specify a new external repository or select an external repository that is preconfigured.
5. Under Additional properties, click **Performance**.

When you finish adding or updating your federated repository configuration, go to the **Security > Global security** panel and click **Apply** to validate the changes.

Limit search time:

Specifies the timeout value in milliseconds for a Lightweight Directory Access Protocol (LDAP) server to respond before stopping a request.

Information	Value
Data type:	Integer
Units:	Milliseconds
Default:	0
Range:	Equal to or greater than 0. A value of 0 specifies that no search time limit exists.

Limit search returns:

Specifies the maximum number of entries that are returned in a search result.

Information	Value
Data type:	Integer
Units:	Entries
Default:	0
Range:	Equal to or greater than 0. A value of 0 specifies that no search return limit exists.

Use connection pooling:

Specifies whether to utilize the connection pooling function, which is provided in the Software Development Kit (SDK).

Connection pooling is maintained by the Java run time. It is configured by system properties.

Information	Value
Default:	Disabled
Range:	Enabled or Disabled

Enable context pool:

Specifies whether context pooling is enabled to the LDAP server. To improve performance, use the context pool in combination with connection pooling.

Information	Value
Default:	Enabled
Range:	Enabled or Disabled

Initial size:

Specifies the number of context instances in the pool when the pool is initially created by the LDAP repository.

Information	Value
Data type:	Integer
Default:	1
Range:	1 to 50

Preferred size:

Specifies the preferred number of context instances that the context pool maintains. Both in-use and idle context instances contribute to this number.

Information	Value
Data type:	Integer
Default:	3
Range:	0 to 100

Maximum size:

Specifies the maximum number of context instances that can be maintained concurrently by the context pool. Both in-use and idle context instances contribute to this number.

When the pool size reaches the maximum size, no new context instances can be created for a new request. The new request is blocked until a context instance is released or removed. The request periodically checks for context instances that are available in the pool. A request for a pooled context instance uses an existing pooled and idle context instance or a newly created pooled context instance.

A maximum pool size of 0 indicates that the context pool can maintain an infinite number of context instances.

Information	Value
Data type:	Integer
Default:	0

Context pool times out:

Specifies the number of seconds for the context pool to time out and remove idle context instances.

A timeout value of 0 indicates that the context pool does not time out context instances.

Information	Value
Data type:	Integer
Default:	0

Cache the attributes:

Specifies whether to cache the attributes that are returned from the LDAP server.

Information	Value
Default:	Enabled
Range:	Enabled or Disabled

Cache size:

Specifies the maximum size of the cache.

Information	Value
Data type:	Integer
Default:	4000
Range:	Equal to or greater than 100

Cache times out:

Specifies the maximum number of seconds that the cached search results can stay in the cache.

A timeout value of 0 indicates that the cached search results stay in the cache until update operations are made.

Information	Value
Data type:	Integer
Units:	Seconds
Default:	1200
Range:	Equal to or greater than 0

Distribution policy:

Specifies the distribution policy for the cache in a clustered environment, which is one of the following:

Not shared
Sends out new entries, both ID and data, and updates to those entries.

Push Requests data from other servers in the cluster when that data is not locally present.

Push and pull
Sends out IDs for new entries and requests from other servers in the cluster entries for IDs that were previously broadcast. The dynamic cache always sends out cache entry invalidations.

Cache the search results:

Specifies whether to cache the search results that are returned from the LDAP server.

Information	Value
Default:	Enabled
Range:	Enabled or Disabled

Cache size:

Specifies the maximum size of the cache.

Information	Value
Data type:	Integer
Default:	2000
Range:	Equal to or greater than 100

Cache times out:

Specifies the maximum number of seconds that the cached search results can stay in the cache.

A timeout value of 0 indicates that the cached search results stay in the cache until update operations are made.

Information	Value
Data type:	Integer
Units:	Seconds
Default:	600
Range:	Equal to or greater than 0

Distribution policy:

Specifies the distribution policy for the cache in a clustered environment, which is one of the following:

Not shared
Sends out new entries, both ID and data, and updates to those entries.

Push Requests data from other servers in the cluster when that data is not locally present.

Push and pull
Sends out IDs for new entries and requests from other servers in the cluster entries for IDs that were previously broadcast. The dynamic cache always sends out cache entry invalidations.

Using custom adapters for federated repositories

When the custom adapters for federated repositories are part of the default realm, the users and groups can be managed using wsadmin commands or the administrative console.

About this task

If custom adapters for federated repositories are part of the default realm, you use the administrative console to manage the users and groups in the realm.

Note: The default parent for PersonAccount and Group entities needs to be the same as the base entry of the custom adapter.

To view this administrative console page, complete the following steps:

- In the administrative console, click **Security > Global security**.
- Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**. To configure for a specific domain in a multiple security domain environment, click **Security domains >** ***domain_name***. Under Security Attributes, expand User Realm, and click **Customize** for this domain. Select the Realm type as **Federated repositories** and then click **Configure**.
- Under Additional properties, click **Supported entity types**.

You must configure the supported entity types before you can manage this account with Users and Groups in the administrative console. The Base entry for the default parent determines the repository location where entities of the specified type are placed on write operations by user and group management.

Procedure

1. In the administrative console, click **Users and Groups** to access users and groups panel. To manage users and groups for a specific domain in a multiple security domain environment, click **Security > Global security > Security Domains >** ***domain_name***. Under Security Attributes, expand User Realm, and click **Customize for this domain**. Select the Realm type as **Federated repositories**. Click **Apply** and **Save** to the master configuration. On Security domains panel that appears, click the ***domain_name*** again to go to the domain configuration panel. The links to manage users and groups for a specific domain are displayed only after you save the federated repositories configuration for the domain.
2. Click **Manage Groups** to test the basic functions of the custom adapter with respect to custom adapters for federated repositories.
3. Click **Manage Users** to test the basic functions of the custom adapter with respect to custom adapters for federated repositories.

   Note: You must restart the server or dmgr if the federated repository has changed before using the Manage Users option. Otherwise, user or group changes made to the repository could be lost after restart.

Results

After completing these steps, you will have ensured that the custom adapter is being used properly.

What to do next

Adjustments to the custom adapter can be made by using the wsadmin tool to make configuration changes. See “Configuring custom adapters for federated repositories using wsadmin” on page 312 for more details.

Sample custom adapters for federated repositories examples:

Out of the box adapters for federated repositories provide File, LDAP, and Database adapters for your use. These adapters implement the com.ibm.wsspi.wim.Repository software programming interface (SPI). A virtual member manager custom adapter needs to implement the same SPI.

Developing custom adapters for federated repositories

Out of the box adapters for federated repositories provide File, LDAP and Database adapters for your use. All these adapters implement the `com.ibm.wsspi.wim.Repository` SPI. See the `com.ibm.wsspi.wim.Repository` SPI for more information. As you develop a virtual member manager custom adapter, you need to implement the same SPI.

Custom adapters for federated repositories must not depend on any WebSphere Application Server components, such as data sources and enterprise beans. These WebSphere Application Server components require that security is initialized and enabled prior to startup. If your implementation of custom adapters for federated repositories needs to use data sources to connect to a database, you need to use Java database connectivity (JDBC) to make the connection during server startup. Then, at a later time, switch to using the data sources when the data source is available.

There are examples of suggested behavior and requirements of custom adapters for federated repositories that you can find in the sample code.

A sample custom adapter for federated repositories

A sample custom adapter implementation has been provided as an example. The custom adapter is based on file repository. The sample source code and class files are bundled in vmmsampleadapter.jar. The `vmmsampleadapter.jar` can be downloaded at this location: http://www.ibm.com/developerworks/websphere/downloads/samples/vmmsampleadapter.html.

Contents of the vmmsampleadapter.jar file are as follows:

- Class files for the sample adapter:
  - `com/ibm/ws/wim/adapter/sample/AbstractAdapterImpl.class`
  - `com/ibm/ws/wim/adapter/sample/SampleFileAdapter.class`
  - `com/ibm/ws/wim/adapter/sample/XPathHelper.class`
- Source code for the sample adapter:
  - `src/com/ibm/ws/wim/adapter/sample/AbstractAdapterImpl.java`
  - `src/com/ibm/ws/wim/adapter/sample/SampleFileAdapter.java`
  - `src/com/ibm/ws/wim/adapter/sample/XPathHelper.java`

Note: The sample files should not be used in the production environment. You should make a copy of these files, rename them, and update them based on your specific adapter implementation. Refer to the Javadoc in the source code for more information.

com/ibm/ws/wim/sample/adapter/AbstractAdapterImpl.java
Provides an abstract implementation class which handles most of the repository independent internal operations for the adapter and defines some simple abstract methods that should be implemented by the custom adapter. For most cases, you may not need to change this file.

com/ibm/ws/wim/sample/adapter/SampleFileAdapter.java
Extends from the AbstractAdapterImpl class and implements the abstracts method. This class implements the abstract methods using file as the repository. Adapter providers can use this class as a reference to implement these methods specific to their adapters.

com/ibm/ws/wim/sample/adapter/XPathHelper.java
Defines a helper class to parse the XPath search expression and build the search tree. This helper class also contains the method to evaluate the search expression. If your repository supports a search expression, then you need to convert XPath expression to an expression that your repository can process and let your repository evaluate the expression. This helper class evaluates the search expression based on the use of dataobjects. You can overwrite the `evaluate()` method to perform the evaluation using other objects, such as **java.util.Map**.

Some utility classes have been provided to help adapter providers. Most of these utility methods are used in the sample adapter. Refer to the Javadoc information for more details.

Establishing custom adapters for federated repositories

Out of the box adapters for federated repositories provide File, LDAP, and Database adapter for your use. These adapters implement the com.ibm.wsspi.wim.Repository software programming interface (SPI). Custom adapters for federated repositories need to implement the same SPI.

Before you begin

Refer to the Repository SPI implementation information in the related references for information about the custom adapters for federated repositories SPI.

Refer to the sample custom adapter code that is available in the `vmmsampleadapter.jar` file. The JAR file contains the sample customer adapter code in the `src/` directory. The `vmmsampleadapter.jar` can be downloaded at this location: http://www.ibm.com/developerworks/websphere/library/samples/vmmsampleadapter.html

Note:

- The sample that is provided is intended to familiarize you with the features of custom adapters for federated repositories and the handling of various types of dataobjects. Do not use this sample in an actual production environment.
- Copy the AbstractAdapterImpl class and rename it before making changes. Make sure that the new name is appropriate for your adapter.

Custom adapters for federated repositories must not depend on any WebSphere Application Server components, such as data sources and enterprise beans. These WebSphere Application Server components require that security is initialized and enabled prior to startup. If your implementation of the virtual member manager custom adapter needs to use data sources to connect to a database, you need to use Java database connectivity (JDBC) to make the connection during server startup. Then, at a later time, switch to using the data sources when the data source is available.

Procedure

1. Build your implementation.

   Note: EMF JAR files contain version number in their names, such as v200607270021. Make sure to change the version number to reflect your installation.

   To compile your code, you need the following JAR files in the classpath:

- *app_server_root*/plugins/com.ibm.ws.runtime_6.1.0.jar
- *app_server_root*/plugins/org.eclipse.emf.commonj.sdo_2.1.0.v200607270021.jar
- *app_server_root*/plugins/org.eclipse.emf.ecore_2.2.1.v200607270021.jar
- *app_server_root*/plugins/org.eclipse.emf.common_2.2.1.v200607270021.jar
- *app_server_root*/plugins/org.eclipse.emf.ecore.xmi_2.2.0.v200607270021.jar
- *app_server_root*/plugins/org.eclipse.emf.ecore.sdo_2.2.0.v200607270021.jar

Here is an example:

```
app_server_root/java/bin/javac -classpath
app_server_root/plugins/com.ibm.ws.runtime_6.1.0.jar;
app_server_root/plugins/org.eclipse.emf.commonj.sdo_2.1.0.
v200607270021.jar;app_server_root
/plugins/org.eclipse.emf.ecore_2.2.1.v200607270021.jar;
app_server_root/plugins/org.eclipse.emf.common_2.2.1.
v200607270021.jar;app_server_root/plugins/org.
eclipse.emf.ecore.xmi_2.2.0.v200607270021.jar;
app_server_root/plugins/org.eclipse.emf.ecore.sdo_2.2.0.v200607270021.jar
your_implementation_file.java
```

2. Copy the generated class files or the packaged JAR file to the product classpath. The preferred location is the *app_server_root*/lib/ext directory. This should be copied to the classpaths of all the product processes (cell and all NodeAgents).
3. Configure your custom adapter by following the steps in “Configuring custom adapters for federated repositories using wsadmin” on page 312.
4. Test your custom adapter by following the steps in “Using custom adapters for federated repositories” on page 308

What to do next

“Configuring custom adapters for federated repositories using wsadmin” on page 312 provides details about configuring your custom adapter with the wsadmin tool.

Adding a custom repository to a federated repositories configuration

Follow this task to add a custom repository under federated repositories.

Procedure

1. In the administrative console, click **Security > Global security**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**. To configure for a specific domain in a multiple security domain environment, click **Security domains > *domain_name***. Under Security Attributes, expand **User Realm**, and click **Customize for this domain**. Select the Realm type as **Federated repositories** and then click **Configure**.
3. Enter the name of the realm in the **Realm name** field. You can change the existing realm name.
4. Enter the name of the primary administrative user in the **Primary administrative user name** field, for example, adminUser.
5. Leave the **Ignore case for authorization** option selected.
6. Leave the **Allow operations if some of the repositories are down** option cleared.
7. Optional: In a multiple security domain environment, select **Use global schema for model** option to indicate that the global schema option is enabled for the data model. Global schema refers to the schema of the admin domain.
8. Click **Add base entry to realm**.
9. Enter a distinguished name for the realm base entry in the **Distinguished name of a base entry that uniquely identifies this set of entries in the realm** field.

10. Enter the distinguished name of the base entry within the repository in the **Distinguished name of a base entry in this repository** field.
11. Click **Add > Custom repository**.
12. Specify the required details for the new custom repository:

    Repository identifier
    Specifies a unique identifier for the repository. This identifier uniquely identifies the repository within the cell.

    Repository adapter class name
    Specifies the implementation class name for the custom repository adapter, for example, com.ibm.ws.wim.adapter.sample.SampleAdapter.

    Login properties
    Specifies the property names to use to log into the application server.

    Custom properties
    Specifies arbitrary name and value pairs of data. The name is a property key and the value is a string value that can be used to set internal system configuration properties.
13. Click **OK** and **Save** to the master configuration.

Results

After completing these steps, your new configuration under Federated repositories includes a custom repository.

Configuring custom adapters for federated repositories using wsadmin

You can use the Jython or Jacl scripting language with the wsadmin tool to define custom adapters in the federated repositories configuration file.

Before you begin

Shut down the WebSphere Application Server and the wsadmin command window.

About this task

Use the following steps to add a custom adapter to any federated repositories configuration file and to any realm defined within the configuration file.

The following examples use the **SampleFileRepository** repository as the identifier for the custom repository.

Note: For additional information about the commands to use for this topic, see the IdMgrRepositoryConfig command group for the AdminTask object topic.

Procedure

1. Enter the following command to start the wsadmin tool:

   ```
   wsadmin –conntype none
   ```
2. Use the **createIdMgrCustomRepository** command to add a custom repository and specify the adapter class.

   The following example configures a custom repository to use the **com.ibm.ws.wim.adapter.sample.SampleFileAdapter** class and sets the **SampleFileRepository** repository as the identifier.

   Using Jython:

   ```
   AdminTask.createIdMgrCustomRepository('-id SampleFileRepository
    -adapterClassName com.ibm.ws.wim.adapter.sample.SampleFileAdapter')
   ```

Using Jacl:

```
createIdMgrCustomRepository {-id SampleFileRepository
 -adapterClassName com.ibm.ws.wim.adapter.sample.SampleFileAdapter}
```

3. Copy the `vmmsampleadapter.jar` file that is provided to `app_server_root/lib`.
4. Disable paging in the common repository configuration. Set the supportPaging parameter for the **updateIdMgrRepository** command to false to disable paging.

   Note: You must perform this step because the sample adapter does not support paging.

   The following examples use the **SampleFileRepository** repository as the identifier for the custom repository.

   Using Jython:

```
AdminTask.updateIdMgrRepository('-id SampleFileRepository -supportPaging false')
```

   Using Jacl:

```
$AdminTask updateIdMgrRepository {-id SampleFileRepository -supportPaging
false}
```

   Note: A warning will appear until the configuration of the sample repository is complete.
5. Add the necessary custom properties for the adapter. Use the **setIdMgrCustomProperty** command repeatedly to add multiple properties. Use this command once per property to add multiple properties to your configuration. You must use both the **name** and **value** parameters to add the custom property for the specified repository. For example, to add a custom property of **fileName**, enter the following command.

   Using Jython:

```
AdminTask.setIdMgrCustomProperty('-id SampleFileRepository -name fileName
-value "c:\sampleFileRegistry.xml"')
```

   Using Jacl:

```
$AdminTask setIdMgrCustomProperty {-id SampleFileRepository -name fileName
-value "c:\sampleFileRegistry.xml"}
```

6. Add a base entry to the adapter configuration. Use the **addIdMgrRepositoryBaseEntry** command to specify the name of the base entry for the specified repository. For example:

   Using Jython:

```
AdminTask.addIdMgrRepositoryBaseEntry('-id SampleFileRepository -name
o=sampleFileRepository')
```

   Using Jacl:

```
$AdminTask addIdMgrRepositoryBaseEntry {-id SampleFileRepository -name
o=sampleFileRepository}
```

7. Use the **addIdMgrRealmBaseEntry** command to add the base entry to the realm, which will link the realm with the repository:

   Using Jython:

```
AdminTask.addIdMgrRealmBaseEntry('-name defaultWIMFileBasedRealm -baseEntry o=sampleFileRepository')
```

   Using Jacl:

```
$AdminTask addIdMgrRealmBaseEntry {-name defaultWIMFileBasedRealm -baseEntry o=sampleFileRepository}
```

8. Save your configuration changes. Enter the following commands to save the new configuration and close the wsadmin tool.

   Using Jython:

```
AdminConfig.save()
exit
```

   Using Jacl:

```
$AdminConfig save
exit
```

   The following example displays the complete text of the newly-revised `wimconfig.xml` file:

   Note: The federated repositories configuration file, `wimconfig.xml`, is located in the *app_server_root*`/profiles/`*profile_name*`/config/cells/`*cell_name*`/wim/config` directory.

```
<!--
  Begin Copyright

  Licensed Materials - Property of IBM

  virtual member manager

  (C) Copyright IBM Corp. 2005 All Rights Reserved.

  US Government Users Restricted Rights - Use, duplication or
  disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

  End Copyright
-->
<sdo:datagraph xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:config="http://www.ibm.com/websphere/wim
/config" xmlns:sdo="commonj.sdo">
  <config:configurationProvider maxPagingResults="500" maxSearchResults="4500"
maxTotalPagingResults="1000"
pagedCacheTimeOut="900" pagingEntityObject="true" searchTimeOut="600000">
    <config:dynamicModel xsdFileName="wimdatagraph.xsd"/>
    <config:supportedEntityTypes defaultParent="o=defaultWIMFileBasedRealm" name="Group">
      <config:rdnProperties>cn</config:rdnProperties>
    </config:supportedEntityTypes>
    <config:supportedEntityTypes defaultParent="o=defaultWIMFileBasedRealm" name="OrgContainer">
      <config:rdnProperties>o</config:rdnProperties>
      <config:rdnProperties>ou</config:rdnProperties>
      <config:rdnProperties>dc</config:rdnProperties>
      <config:rdnProperties>cn</config:rdnProperties>
    </config:supportedEntityTypes>
    <config:supportedEntityTypes defaultParent="o=defaultWIMFileBasedRealm" name="PersonAccount">
      <config:rdnProperties>uid</config:rdnProperties>
    </config:supportedEntityTypes>
    <config:repositories xsi:type="config:FileRepositoryType" adapterClassName="com.ibm.
    ws.wim.adapter.file.was.FileAdapter"
id="InternalFileRepository" supportPaging="false" supportSorting="false" messageDigestAlgorithm="SHA-1">
      <config:baseEntries name="o=defaultWIMFileBasedRealm"/>
    </config:repositories>
    <config:repositories adapterClassName="com.ibm.ws.wim.adapter.sample.SampleFileAdapter"
    id="SampleFileRepository">
      <config:CustomProperties name="fileName" value="c:\sampleFileRegistry.xml"/>
      <config:baseEntries name="o=sampleFileRepository"/>
    </config:repositories>
    <config:realmConfiguration defaultRealm="defaultWIMFileBasedRealm">
      <config:realms delimiter="@" name="defaultWIMFileBasedRealm" securityUse="active">
        <config:participatingBaseEntries name="o=defaultWIMFileBasedRealm"/>
        <config:participatingBaseEntries name="o=sampleFileRepository"/>
        <config:uniqueUserIdMapping propertyForInput="uniqueName" propertyForOutput="uniqueName"/>
        <config:userSecurityNameMapping propertyForInput="principalName" propertyForOutput="principalName"/>
        <config:userDisplayNameMapping propertyForInput="principalName" propertyForOutput="principalName"/>
        <config:uniqueGroupIdMapping propertyForInput="uniqueName" propertyForOutput="uniqueName"/>
        <config:groupSecurityNameMapping propertyForInput="cn" propertyForOutput="cn"/>
        <config:groupDisplayNameMapping propertyForInput="cn" propertyForOutput="cn"/>
      </config:realms>
    </config:realmConfiguration>
</config:configurationProvider></sdo:datagraph>
```

9. Restart the application server.

Configuring the user registry bridge for federated repositories using wsadmin scripting

The user registry bridge is configured like other custom adapters. You can use the Jython or Jacl scripting language with the wsadmin scripting tool to define the user registry bridge in the federated repositories configuration.

Before you begin

Shut down WebSphere Application Server and the wsadmin command window.

Important: If you are migrating from the stand-alone user registry on the local operating system to federated repositories on the local operating system, you must first configure the current user registry under federated repositories. For more information, see Managing the realm in a federated repository configuration.

Authorization failures might occur if users or groups are mapped to roles before migration and you use those users or groups after migrating to user registry bridge. This situation occurs because the mapping contains registry-specific information. After migration, re-map the users or groups to avoid authorization failures.

About this task

For additional information about the commands to use for this topic, see **IdMgrRepositoryConfig** command group for the AdminTask object.

Use the following steps to add a user registry bridge to any federated repositories configuration and to any realm that is defined within the configuration.

Procedure

1. Start the wsadmin scripting tool. You can use the following command to start the wsadmin scripting tool:

```
wsadmin –conntype none
```

2. Use the **createIdMgrCustomRepository** command to add a new repository configuration for the user registry bridge.

   The following example configures a custom repository to use the com.ibm.ws.wim.adapter.urbridge.URBridge class and sets urbcustom as the identifier:

   Using Jython:

```
AdminTask.createIdMgrCustomRepository('-id urbcustom
-adapterClassName com.ibm.ws.wim.adapter.urbridge.URBridge')
```

   Using Jacl:

```
$AdminTask createIdMgrCustomRepository {-id urbcustom
-adapterClassName com.ibm.ws.wim.adapter.urbridge.URBridge}
```

   gotcha: The user registry bridge handles requests to one user registry only. Therefore, if you define multiple repositories, each user registry implementation must have a separate instance of the user registry bridge and you must define each implementation as a separate repository with a unique repository ID..

3. Optional: Add the necessary registry-specific properties as custom properties. Use the **setIdMgrCustomProperty** command repeatedly to add multiple properties. Use this command once per property to add multiple properties to your configuration. You must use both the name and value parameters to add the custom property for the specified repository. For example, to add a custom property of uniqueUserIdProperty, enter the following command:

   Using Jython:

```
AdminTask.setIdMgrCustomProperty('-id urbcustom
-name uniqueUserIdProperty -value "uniqueId"')
```

   Using Jacl:

```
$AdminTask setIdMgrCustomProperty {-id urbcustom
-name uniqueUserIdProperty -value "uniqueId"}
```

   To configure the user registry bridge to use a custom user registry, you must add the registryImplClass property and specify the exact registry implementation class. For example, specify com.xyz.abc.MyCustomRegistry as the value for the property.

   To configure the user registry bridge to use the local operating system user registry, do not specify the registryImplClass property. The user registry bridge identifies the underlying user registry implementation that is provided by WebSphere Application Server for the local operating system.

   You can set other optional properties as custom properties to define the mapping between federated repository properties and user registry properties, such as uniqueUserIdProperty, userSecurityNameProperty, userDisplayNameProperty, uniqueGroupIdProperty, groupSecurityNameProperty, and groupDisplayNameProperty. For more information about the available custom properties and their default values, see Security custom properties. To override any of these properties at the user registry level, configure the property as a custom property.

   gotcha: The mapping between a federated repository property and user registry property is one-to-one. You can map only one federated repository property to a user registry property.

4. Add a base entry to the user registry bridge configuration. Use the **addIdMgrRepositoryBaseEntry** command to specify the name of the base entry for the specified repository. For example:

   Using Jython:

   ```
   AdminTask.addIdMgrRepositoryBaseEntry('-id urbcustom
   -name o=custom')
   ```

   Using Jacl:

   ```
   $AdminTask addIdMgrRepositoryBaseEntry {-id urbcustom
   -name o=custom}
   ```

5. Use the **addIdMgrRealmBaseEntry** command to add the base entry to the realm, which will link the realm with the repository.

   Note: The default realm name is defaultWIMFileBasedRealm. If this realm name was previously renamed, use the new realm name instead of defaultWIMFileBasedRealm. For example, to ensure consistency, you can set the realm name of the federated repository configuration to be the same name as the local operating system user registry as specified in the `security.xml` file. For information about how to set the realm name, see Realm configuration settings.

   Use the following command:

   Using Jython:

   ```
   AdminTask.addIdMgrRealmBaseEntry('-name defaultWIMFileBasedRealm
   -baseEntry o=custom')
   ```

   Using Jacl:

   ```
   $AdminTask addIdMgrRealmBaseEntry {-name defaultWIMFileBasedRealm
   -baseEntry o=custom}
   ```

6. Save your configuration changes. Enter the following commands to save the new configuration and close the wsadmin scripting tool:

   Using Jython:

   ```
   AdminConfig.save()
   exit
   ```

   Using Jacl:

   ```
   $AdminConfig save
   exit
   ```

7. Restart the application server.

Results

The following code is an example of a basic configuration in the `wimconfig.xml` file for a user registry bridge accessing a custom user registry:

```
<config:repositories adapterClassName="com.ibm.ws.wim.adapter.urbridge.URBridge" id="urbcustom">
<config:baseEntries name="o=custom"/>
  <config:CustomProperties name="registryImplClass" value="com.ibm.registry.impl.FileRegistrySample"/>
  <config:CustomProperties name="usersFile" value="${USER_PROPS}"/>
  <config:CustomProperties name="groupsFile" value="${GROUP_PROPS}"/>
</config:repositories>
```

In the previous example, the *${USER_PROPS}* and *${GROUP_PROPS}* variables are used to define the values of the custom properties.

You can use variables to define custom properties. However, these variables are resolved only in the WebSphere Application Server connected mode. For information about how to define environment variables, see Creating, editing, and deleting WebSphere variables.

User registry bridge for federated repositories:

The user registry bridge is a read-only adapter that provides an interface between federated repositories and an underlying user registry implementation, which can be either a local operating system user registry or a custom user registry implementation.

The user registry bridge enables IBM WebSphere Application Server applications to use your user registry implementation. It can work with any user registry that implements the com.ibm.websphere.security.UserRegistry interface, without knowing the details of its implementation. This capability makes the bridge versatile and allows it to connect to, and use, various registries.

The user registry bridge allows access to the same repository information without any platform-specific implementation. Thus, it eliminates the need to have a specialized user registry bridge for each operating system.

You can federate and configure the local operating system user registry, a custom user registry, or both, as a federated repository. The user registry bridge handles user registry-related requests from federated repositories, makes appropriate calls to the underlying user registry implementation, and returns data that is formatted according to the federated repositories specifications.

Therefore, to use the user registry bridge you must configure your user registry under federated repositories. This configuration can map the properties in the underlying user registry to the properties for the federated repository. You can also configure any user registry specific information, if required. For information about how to configure the user registry bridge, see Configuring the user registry bridge for federated repositories using wsadmin scripting.

The following figure illustrates the difference between configuring a federated repository user registry with and without the user registry bridge.

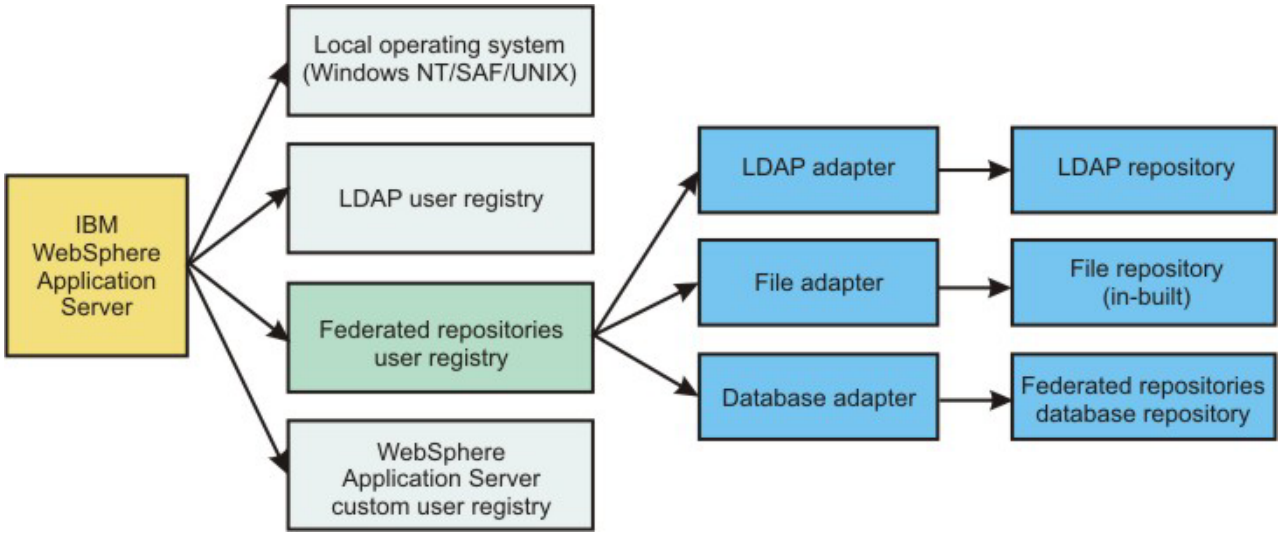


Figure 5. Configuring a federated repository user registry without the user registry bridge

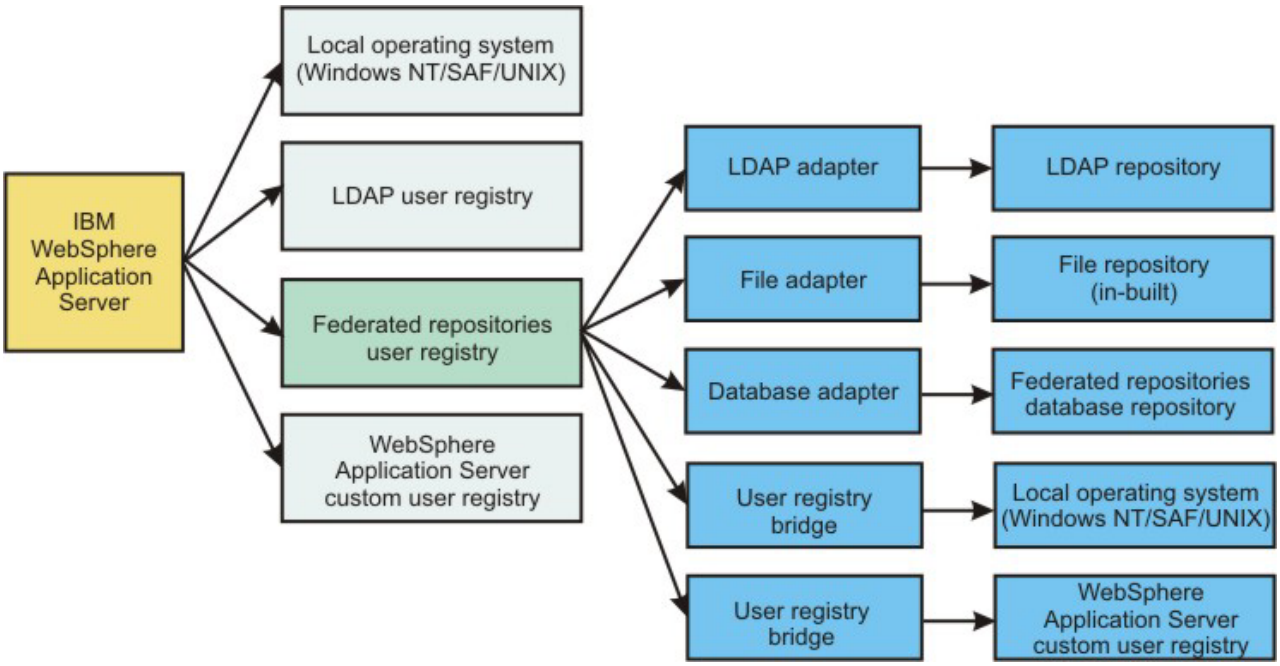


Figure 6. Configuring a federated repository user registry with the user registry bridge

As shown in the previous figure, using the same adapter, which is the user registry bridge, you can configure multiple user registries under federated repositories. For example, you can configure a local operating system user registry and one or more custom user registries.

Limitations

The following limitations exist:

- You can use the user registry bridge only for read-only operations, such as authentication and search functions. You cannot perform write operations such as create, delete, or modify users and groups. An attempt to perform write operations results in an exception, which notifies the user that the operation is not supported by the bridge. This limitation exists because the user registry bridge does not have direct access to the repository. Instead, the bridge uses an underlying existing user registry implementation that is read-only; hence, it might not be able to fulfill requests for certain properties that exist in the federated repositories.
- The user registry bridge does not support a stand-alone Lightweight Directory Access Protocol (LDAP) user registry. LDAP repositories are supported as a standard federated repositories adapter with read and write capabilities.
- Some of the properties that are placed in control data objects are not relevant to the user registry bridge as they are not applicable in the underlying repository.
  - The properties ignored for GroupMembershipControl and GroupMemberControl data objects are searchBases, timeLimit, treeView, expression, and level.
  - The properties ignored for SearchControl data objects are searchBases and timeLimit. The property part of the expression, such as uid and mail, is ignored as you can search WebSphere Application Server user registry entities with security names only. The expression is parsed to get the entity type and the pattern with which the search must be performed.

Supported user registries

WebSphere Application Server applications can access the user registry properties of the following user registry implementations as a read-only repository:

- Local operating system user registry

- Custom user registry

Configuring Lightweight Directory Access Protocol entity types in a federated repository configuration

Follow this task to configure Lightweight Directory Access Protocol (LDAP) entity types in a federated repository configuration.

Procedure

1. In the administrative console, click **Security > Global security**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**. To configure for a specific domain in a multiple security domain environment, click **Security domains > *domain_name***. Under Security Attributes, expand User Realm, and click **Customize for this domain**. Select the Realm type as Federated repositories and then click **Configure**.
3. Under Related items, click **Manage repositories**.
4. Click **Add** to specify a new external repository or select an external repository that is preconfigured. During LDAP configuration, based on the selected LDAP server type, some defaults and mappings are set in the configuration. When the selected LDAP server type is custom, no default is set, and you must set all of the mappings manually. To avoid setting all of the mappings manually, choose a non-custom LDAP server type (for example, IBM Directory Server or SunOne) which matches closely to your LDAP server.

   Note:

   - If you click **Add** to specify a new external repository, you must first complete the required fields and click **Apply** before you can proceed to the next step.
   - If you decide to use a custom LDAP server type, you must use the command-line interface to create the entity types. Read about IdMgrRepositoryConfig command group for the AdminTask object for more information.

     After you create the entity types, you can use the administrative console to modify these entities. You cannot use the administrative console to create entity types for a custom LDAP server type.
5. Under Additional properties, click **LDAP entity types**.
6. View the entity types that are supported by the member repositories, or select an entity type to view or change its configuration properties.
7. Supply the object classes that are mapped to this entity type in the Object classes field. LDAP entries that contain one or more of the object classes belong to this entity type.
8. Supply the search bases that are used to search this entity type. The search bases specified must be subtrees of the base entry in the repository. For example, you can specify the following search bases, where `o=ibm,c=us` is the base entry in the repository:

   `o=ibm,c=us` or `cn=users,o=ibm,c=us` or `ou=austin,o=ibm,c=us`

   In the preceding example, you cannot specify search bases `c=us` or `o=ibm,c=uk`.

   Delimit multiple search bases with a semicolon (;). For example:

   `ou=austin,o=ibm,c=us;ou=raleigh,o=ibm,c=us`
9. Supply the LDAP search filter that is used to search this entity type.

   For example, use `(objectclass=ePerson)` to search for users or `(|(objectclass=groupOfNames)(objectclass=groupOfUniqueNames)` to search for groups in an external LDAP repository.

   If a search filter is not specified, the object classes and the relative distinguished name (RDN) properties are used to generate the search filter. For information on RDN properties, see “Configuring supported entity types in a federated repository configuration” on page 295.

Results

After completing these steps, LDAP entity types are configured for your LDAP repository.

What to do next

1. After configuring the federated repositories, click **Security > Global security** to return to the Global security panel. Verify that Federated repositories is identified in the Current realm definition field. If Federated repositories is not identified, select **Federated repositories** from the Available realm definitions field and click **Set as current**. To verify the federated repositories configuration, click **Apply** on the Global security panel. If Federated repositories is not identified in the Current realm definition field, your federated repositories configuration is not used by WebSphere Application Server.
2. If you are enabling security, complete the remaining steps as specified in "Enabling security for the realm" on page 84. As the final step, validate this setup by clicking **Apply** in the Global security panel.
3. Save, stop, and restart all the product servers (deployment managers, nodes, and Application Servers) for changes in this panel to take effect. If the server comes up without any problems, the setup is correct.

Lightweight Directory Access Protocol entity types collection:

Use this page to list Lightweight Directory Access Protocol (LDAP) entity types that are supported by the member repositories or to select an LDAP entity type to view or change its configuration properties.

To view this administrative console page, complete the following steps:

1. In the administrative console, click **Security > Global security**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**. To configure for a specific domain in a multiple security domain environment, click **Security domains > *domain_name***. Under Security Attributes, expand User Realm, and click **Customize for this domain**. Select the Realm type as **Federated repositories** and then click **Configure**.
3. Under Related items, click **Manage repositories**.
4. Click **Add** to specify a new external repository or select an external repository that is preconfigured.
5. Under Additional properties, click **LDAP entity types**.

When you finish adding or updating your federated repository configuration, go to the **Security > Global security** panel and click **Apply** to validate the changes.

Entity type:

Specifies the entity type name.

Object classes:

Specifies the object classes that are mapped to this entity type. LDAP entries that contain one or more of the object classes belong to this entity type.

You cannot map multiple entity types to the same LDAP object class.

Search bases:

Specifies the search bases that are used to search this entity type.

The search bases specified must be subtrees of the base entry in the repository. For example, you can specify the following search bases, where `o=ibm,c=us` is the base entry in the repository:

`o=ibm,c=us` or `cn=users,o=ibm,c=us` or `ou=austin,o=ibm,c=us`

In the preceding example, you cannot specify search bases `c=us` or `o=ibm,c=uk`.

Delimit multiple search bases with a semicolon (;). For example:

```
ou=austin,o=ibm,c=us;ou=raleigh,o=ibm,c=us
```

Search filter:

Specifies the LDAP search filter that is used to search this entity type.

For example, use `(objectclass=ePerson)` to search for users or `(&(cn=%v)(|(objectclass=groupOfNames)(objectclass=groupOfUniqueNames)))` to search for groups in an external LDAP repository.

If a search filter is not specified, the object classes and the relative distinguished name (RDN) properties are used to generate the search filter.

Lightweight Directory Access Protocol attributes collection:

Use this page to add, modify, or delete the configuration of supported, unsupported, and external LDAP attributes in a federated repositories configuration.

To view this administrative console page, complete the following steps:

1. In the administrative console, click **Security > Global security**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**. To configure for a specific domain in a multiple security domain environment, click **Security domains > *domain_name***. Under Security Attributes, expand User Realm, and click **Customize for this domain**. Select the Realm type as Federated repositories and then click **Configure**.
3. Under Related items, click **Manage repositories**.
4. Click **Add > LDAP repository** to specify a new external repository or select an external repository that is preconfigured.
5. Under Additional properties, click **LDAP attributes**.
6. To add a new LDAP attribute configuration, click **Add** and select **Supported** , **Unsupported**, or **External**.
7. To modify an existing configuration, click the **Name/Property Name** link and modify the details in the panel that appears.
8. To delete an existing configuration, select the checkbox beside the Name/Property Name and click **Delete**.

When you finish adding or updating your federated repository configuration, go to the **Security > Global security** panel and click **Apply** to validate the changes.

Supported:

Specifies the configuration for supported LDAP attributes.

Name Specifies the name of the LDAP attribute used in the repository LDAP adapter.

Property name
Specifies the name of the corresponding federated repository property.

Syntax
Specifies the syntax of the LDAP attribute. The default value is string. For example, the syntax of the unicodePwd LDAP attribute is octetString.

Entity types
Specifies the entity type that applies the attribute mapping.

Default value
Specifies the default value of the LDAP attribute.

Default attribute
Use this parameter to specify the default attribute of the LDAP attribute.

Unsupported:

Specifies the configuration for a federated repository property that the LDAP repository does not support.

Property name
Specifies the name of the federated repository property.

Entity types
Specifies one or more entity types. Use the semicolon (;) as the delimiter to specify multiple entity types.

External:

Specifies the configuration for an LDAP attribute that is used as an external ID in the specified LDAP repository.

Name Specifies the name of the external ID attribute of the LDAP repository.

Syntax
Specifies the syntax of the LDAP attribute. The default value is string. For example, the syntax of the unicodePwd LDAP attribute is octetString.

Entity types
Specifies one or more entity types. Use the semicolon (;) as the delimiter to specify multiple entity types.

Generate value
Specifies whether or not the federated repository should generate the value of the LDAP attribute.

Lightweight Directory Access Protocol entity types settings:

Use this page to configure Lightweight Directory Access Protocol (LDAP) entity types that are supported by the member repositories.

To view this administrative console page, complete the following steps:

1. In the administrative console, click **Security > Global security**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**. To configure for a specific domain in a multiple security domain environment, click **Security domains > *domain_name***. Under Security Attributes, expand User Realm, and click **Customize for this domain**. Select the Realm type as **Federated repositories** and then click **Configure**.
3. Under Related items, click **Manage repositories**.
4. Click **Add** to specify a new external repository or select an external repository that is preconfigured.
5. Under Additional properties, click **LDAP entity types**.
6. Select an entity type to view or change its configuration properties.

When you finish adding or updating your federated repository configuration, go to the **Security > Global security** panel and click **Apply** to validate the changes.

Entity type:

Specifies the entity type.

Object classes:

Specifies the object classes that are mapped to this entity type. LDAP entries that contain one or more of the object classes belong to this entity type.

You cannot map multiple entity types to the same LDAP object class.

Search bases:

Specifies the search bases that are used to search this entity type.

The search bases specified must be subtrees of the base entry in the repository. For example, you can specify the following search bases, where `o=ibm,c=us` is the base entry in the repository:

`o=ibm,c=us` or `cn=users,o=ibm,c=us` or `ou=austin,o=ibm,c=us`

In the preceding example, you cannot specify search bases `c=us` or `o=ibm,c=uk`.

Delimit multiple search bases with a semicolon (;). For example:

```
ou=austin,o=ibm,c=us;ou=raleigh,o=ibm,c=us
```

Search filter:

Specifies the LDAP search filter that is used to search this entity type.

For example, use `(objectclass=ePerson)` to search for users or `(&(cn=%v)(|(objectclass=groupOfNames)(objectclass=groupOfUniqueNames)))` to search for groups in an external LDAP repository.

If a search filter is not specified, the object classes and the relative distinguished name (RDN) properties are used to generate the search filter.

Configuring Lightweight Directory Access Protocol attributes in a federated repository configuration

Follow this task to add, modify, or delete the configuration of supported, unsupported, and external LDAP attributes in a federated repositories configuration.

Procedure

1. In the administrative console, click **Security > Global security**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**. To configure for a specific domain in a multiple security domain environment, click **Security domains > *domain_name***. Under Security Attributes, expand User Realm, and click **Customize for this domain**. Select the Realm type as Federated repositories and then click **Configure**.
3. Under Related items, click **Manage repositories**, and then in the panel that appears, click the ***repository_id*** of the LDAP repository.
4. Under Additional properties, click the **LDAP attributes** link.
5. To add a new LDAP attribute configuration, click **Add** and select one of the following options:
   - Select **Supported** to add a supported LDAP attribute configuration. On the panel that appears, enter the following details:

     Name Specifies the name of the LDAP attribute used in the repository LDAP adapter.

Property name
Specifies the name of the corresponding federated repository property.

Syntax
Specifies the syntax of the LDAP attribute. The default value is string. For example, the syntax of the unicodePwd LDAP attribute is octetString.

Entity types
Specifies the entity type that applies the attribute mapping.

Default value
Specifies the default value of the LDAP attribute.

Default attribute
Use this parameter to specify the default attribute of the LDAP attribute.

- Select **Unsupported** to add a configuration for a federated repository property that the LDAP repository does not support. On the panel that appears, enter the following details:

  Property name
  Specifies the name of the federated repository property.

  Entity types
  Specifies one or more entity types. Use the semicolon (;) as the delimiter to specify multiple entity types.
- Select External to add a configuration for an LDAP attribute that is used as an external ID in the specified LDAP repository. On the panel that appears, enter the following details:

  Name Specifies the name of the external ID attribute of the LDAP repository.

  Syntax
  Specifies the syntax of the LDAP attribute. The default value is string. For example, the syntax of the unicodePwd LDAP attribute is octetString.

  Entity types
  Specifies one or more entity types. Use the semicolon (;) as the delimiter to specify multiple entity types.

  Generate value
  Specifies whether or not the federated repository should generate the value of the LDAP attribute.

6. To modify an existing configuration, click the **Name/Property Name** link and modify the details in the panel that appears.
7. To delete an existing configuration, select the checkbox beside the Name/Property Name and click **Delete**.
8. Click **OK** and **Save** to the master configuration.
9. Restart the application server.

Results

After completing these steps, LDAP attributes are configured in the federated repositories configuration.

Lightweight Directory Access Protocol entity types collection:

Use this page to list Lightweight Directory Access Protocol (LDAP) entity types that are supported by the member repositories or to select an LDAP entity type to view or change its configuration properties.

To view this administrative console page, complete the following steps:

1. In the administrative console, click **Security > Global security**.

2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**. To configure for a specific domain in a multiple security domain environment, click **Security domains** > ***domain_name***. Under Security Attributes, expand User Realm, and click **Customize for this domain**. Select the Realm type as **Federated repositories** and then click **Configure**.
3. Under Related items, click **Manage repositories**.
4. Click **Add** to specify a new external repository or select an external repository that is preconfigured.
5. Under Additional properties, click **LDAP entity types**.

When you finish adding or updating your federated repository configuration, go to the **Security > Global security** panel and click **Apply** to validate the changes.

Entity type:

Specifies the entity type name.

Object classes:

Specifies the object classes that are mapped to this entity type. LDAP entries that contain one or more of the object classes belong to this entity type.

You cannot map multiple entity types to the same LDAP object class.

Search bases:

Specifies the search bases that are used to search this entity type.

The search bases specified must be subtrees of the base entry in the repository. For example, you can specify the following search bases, where `o=ibm,c=us` is the base entry in the repository:

`o=ibm,c=us` or `cn=users,o=ibm,c=us` or `ou=austin,o=ibm,c=us`

In the preceding example, you cannot specify search bases `c=us` or `o=ibm,c=uk`.

Delimit multiple search bases with a semicolon (;). For example:

```
ou=austin,o=ibm,c=us;ou=raleigh,o=ibm,c=us
```

Search filter:

Specifies the LDAP search filter that is used to search this entity type.

For example, use `(objectclass=ePerson)` to search for users or `(&(cn=%v)(|(objectclass=groupOfNames)(objectclass=groupOfUniqueNames)))` to search for groups in an external LDAP repository.

If a search filter is not specified, the object classes and the relative distinguished name (RDN) properties are used to generate the search filter.

Lightweight Directory Access Protocol entity types settings:

Use this page to configure Lightweight Directory Access Protocol (LDAP) entity types that are supported by the member repositories.

To view this administrative console page, complete the following steps:

1. In the administrative console, click **Security > Global security**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**. To configure for a specific domain in a multiple security domain environment, click **Security domains > *domain_name***. Under Security Attributes, expand User Realm, and click **Customize for this domain**. Select the Realm type as **Federated repositories** and then click **Configure**.
3. Under Related items, click **Manage repositories**.
4. Click **Add** to specify a new external repository or select an external repository that is preconfigured.
5. Under Additional properties, click **LDAP entity types**.
6. Select an entity type to view or change its configuration properties.

When you finish adding or updating your federated repository configuration, go to the **Security > Global security** panel and click **Apply** to validate the changes.

Entity type:

Specifies the entity type.

Object classes:

Specifies the object classes that are mapped to this entity type. LDAP entries that contain one or more of the object classes belong to this entity type.

You cannot map multiple entity types to the same LDAP object class.

Search bases:

Specifies the search bases that are used to search this entity type.

The search bases specified must be subtrees of the base entry in the repository. For example, you can specify the following search bases, where `o=ibm,c=us` is the base entry in the repository:

`o=ibm,c=us` or `cn=users,o=ibm,c=us` or `ou=austin,o=ibm,c=us`

In the preceding example, you cannot specify search bases `c=us` or `o=ibm,c=uk`.

Delimit multiple search bases with a semicolon (;). For example:

```
ou=austin,o=ibm,c=us;ou=raleigh,o=ibm,c=us
```

Search filter:

Specifies the LDAP search filter that is used to search this entity type.

For example, use `(objectclass=ePerson)` to search for users or `(&(cn=%v)(|(objectclass=groupOfNames)(objectclass=groupOfUniqueNames)))` to search for groups in an external LDAP repository.

If a search filter is not specified, the object classes and the relative distinguished name (RDN) properties are used to generate the search filter.

Lightweight Directory Access Protocol attributes collection:

Use this page to add, modify, or delete the configuration of supported, unsupported, and external LDAP attributes in a federated repositories configuration.

To view this administrative console page, complete the following steps:

1. In the administrative console, click **Security > Global security**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**. To configure for a specific domain in a multiple security domain environment, click **Security domains > *domain_name***. Under Security Attributes, expand User Realm, and click **Customize for this domain**. Select the Realm type as Federated repositories and then click **Configure**.
3. Under Related items, click **Manage repositories**.
4. Click **Add > LDAP repository** to specify a new external repository or select an external repository that is preconfigured.
5. Under Additional properties, click **LDAP attributes**.
6. To add a new LDAP attribute configuration, click **Add** and select **Supported** , **Unsupported**, or **External**.
7. To modify an existing configuration, click the **Name/Property Name** link and modify the details in the panel that appears.
8. To delete an existing configuration, select the checkbox beside the Name/Property Name and click **Delete**.

When you finish adding or updating your federated repository configuration, go to the **Security > Global security** panel and click **Apply** to validate the changes.

Supported:

Specifies the configuration for supported LDAP attributes.

Name Specifies the name of the LDAP attribute used in the repository LDAP adapter.

Property name
Specifies the name of the corresponding federated repository property.

Syntax
Specifies the syntax of the LDAP attribute. The default value is string. For example, the syntax of the unicodePwd LDAP attribute is octetString.

Entity types
Specifies the entity type that applies the attribute mapping.

Default value
Specifies the default value of the LDAP attribute.

Default attribute
Use this parameter to specify the default attribute of the LDAP attribute.

Unsupported:

Specifies the configuration for a federated repository property that the LDAP repository does not support.

Property name
Specifies the name of the federated repository property.

Entity types
Specifies one or more entity types. Use the semicolon (;) as the delimiter to specify multiple entity types.

External:

Specifies the configuration for an LDAP attribute that is used as an external ID in the specified LDAP repository.

Name Specifies the name of the external ID attribute of the LDAP repository.

Syntax
Specifies the syntax of the LDAP attribute. The default value is string. For example, the syntax of the unicodePwd LDAP attribute is octetString.

Entity types
Specifies one or more entity types. Use the semicolon (;) as the delimiter to specify multiple entity types.

Generate value
Specifies whether or not the federated repository should generate the value of the LDAP attribute.

Configuring group attribute definition settings in a federated repository configuration

Follow this task to configure group definition settings in a federated repository configuration.

Before you begin

Because group attribute definition settings apply only to a Lightweight Directory Access Protocol (LDAP) repository, you must first configure an LDAP repository. For more information, see "Managing repositories in a federated repository configuration" on page 300.

Procedure

1. In the administrative console, click **Security > Global security**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**. To configure for a specific domain in a multiple security domain environment, click **Security domains >** ***domain_name***. Under Security Attributes, expand User Realm, and click **Customize for this domain**. Select the Realm type as **Federated repositories** and then click **Configure**.
3. Under Related items, click **Manage repositories**.
4. Click **Add > LDAP repository** to specify a new external repository or select an external repository that is preconfigured.

   Note: If you click **Add** to specify a new external repository, you must first complete the required fields and click **Apply** before you can proceed to the next step.
5. Under Additional properties, click **Group attribute definition**.
6. Supply the name of the group membership attribute in the Name of group membership attribute field. Only one membership attribute can be defined for each LDAP repository.

   Every LDAP entry should have this attribute to indicate the groups to which this entry belongs. For example, memberOf is the name of the membership attribute that is used in Active Directory. The group membership attribute contains values that reference groups to which this entry belongs. If UserA belongs to GroupA, then the value of the memberOf attribute of UserA should contain the distinguished name of GroupA.

   If your LDAP server does not support the group membership attribute, then do not specify this attribute. The LDAP repository can look up groups by searching the group member attributes, though the performance might be slower.
7. Select the scope of the group membership attribute. The default value is Direct.

   Direct The membership attribute contains direct groups only. Direct groups are the groups that contain the member. For example, if Group1 contains Group2 and Group2 contains User1, then Group2 is a direct group of User1, but Group1 is not a direct group of User1.

   Nested
   The membership attribute contains both direct groups and nested groups.

   All The membership attribute contains direct groups, nested groups, and dynamic members.

Results

After completing these steps, group attribute definition settings are configured for your LDAP repository.

What to do next

1. After configuring the federated repositories, click **Security > Global security** to return to the Global security panel. Verify that Federated repositories is identified in the Current realm definition field. If Federated repositories is not identified, select **Federated repositories** from the Available realm definitions field and click **Set as current**. To verify the federated repositories configuration, click **Apply** on the Global security panel. If Federated repositories is not identified in the Current realm definition field, your federated repositories configuration is not used by WebSphere Application Server.
2. If you are enabling security, complete the remaining steps as specified in "Enabling security for the realm" on page 84. As the final step, validate this setup by clicking **Apply** in the Global security panel.
3. Save, stop, and restart all the product servers (deployment managers, nodes, and Application Servers) for changes in this panel to take effect. If the server comes up without any problems, the setup is correct.

Group attribute definition settings:

Use this page to specify the name of the group membership attribute. Every Lightweight Directory Access Protocol (LDAP) entry includes this attribute to indicate the group to which this entry belongs.

To view this administrative console page, complete the following steps:

1. In the administrative console, click **Security > Global security**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**. To configure for a specific domain in a multiple security domain environment, click **Security domains > *domain_name***. Under Security Attributes, expand User Realm, and click **Customize for this domain**. Select the Realm type as **Federated repositories** and then click **Configure**.
3. Under Related items, click **Manage repositories**.
4. Click **Add > LDAP repository** to specify a new external repository or select an external repository that is preconfigured.
5. Under Additional properties, click **Group attribute definition**.

When you finish adding or updating your federated repository configuration, go to the **Security > Global security** panel and click **Apply** to validate the changes.

Name of group membership attribute:

Specifies the name of the group membership attribute. Only one membership attribute can be defined for each Lightweight Directory Access Protocol (LDAP) repository.

Every LDAP entry should have this attribute to indicate the groups to which this entry belongs. For example, memberOf is the name of the membership attribute that is used in Active Directory. The group membership attribute contains values that reference groups to which this entry belongs. If UserA belongs to GroupA, then the value of the memberOf attribute of UserA should contain the distinguished name of GroupA.

If your LDAP server does not support the group membership attribute, then do not specify this attribute. The LDAP repository can look up groups by searching the group member attributes, though the performance might be slower.

Scope of group membership attribute:

Specifies the scope of the group membership attribute.

Information	Value
Default:	Direct
Range:	**Direct** The membership attribute contains direct groups only. Direct groups are the groups that contain the member. For example, if Group1 contains Group2 and Group2 contains User1, then Group2 is a direct group of User1, but Group1 is not a direct group of User1. **Nested** The membership attribute contains both direct groups and nested groups. **All** The membership attribute contains direct groups, nested groups, and dynamic members.

Configuring member attributes in a federated repository configuration

Follow this task to configure member attributes in a federated repository configuration.

Before you begin

Because member attributes apply only to a Lightweight Directory Access Protocol (LDAP) repository, you must first configure an LDAP repository. For more information, see “Managing repositories in a federated repository configuration” on page 300.

Procedure

1. In the administrative console, click **Security > Global security**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**. To configure for a specific domain in a multiple security domain environment, click **Security domains > *domain_name***. Under Security Attributes, expand User Realm, and click **Customize for this domain**. Select the Realm type as **Federated repositories** and then click **Configure**.
3. Under Related items, click **Manage repositories**.
4. Click **Add > LDAP repository** to specify a new external repository or select an external repository that is preconfigured.

   Note: If you click **Add** to specify a new external repository, you must first complete the required fields and click **Apply** before you can proceed to the next step.
5. Under Additional properties, click **Group attribute definition**.
6. Under Additional properties, click **Member attributes**.
7. Click **New** to specify a new member attribute or **Delete** to remove a preconfigured member attribute.
8. Accept the default, or supply the name of the member attribute in the Name of member attribute field. For example, member and uniqueMember are two commonly used names of member attributes.

   The member attribute is used to store the values that reference members that the group contains. For example, a group type with an object class groupOfNames has a member attribute named member; group type with object class groupOfUniqueNames has a member attribute named uniqueMember. An LDAP repository supports multiple group types if multiple member attributes and their associated group object classes are specified.
9. Supply the object class of the group that uses this member attribute in the Object class field. If this field is not defined, this member attribute applies to all group object classes.
10. Select the scope of the member attribute. The default value is Direct.

    Direct The member attribute contains direct members only. Direct members are members that are

directly contained by the group. For example, if Group1 contains Group2 and Group2 contains User1, then User1 is a direct member of Group2, but User1 is not a direct member of Group1.

Nested
The member attribute contains both direct members and nested members.

All The member attribute contains direct members, nested members, and dynamic members.

Results

After completing these steps, member attributes are configured for your LDAP repository.

What to do next

1. After configuring the federated repositories, click **Security > Global security** to return to the Global security panel. Verify that Federated repositories is identified in the Current realm definition field. If Federated repositories is not identified, select **Federated repositories** from the Available realm definitions field and click **Set as current**. To verify the federated repositories configuration, click **Apply** on the Global security panel. If Federated repositories is not identified in the Current realm definition field, your federated repositories configuration is not used by WebSphere Application Server.
2. If you are enabling security, complete the remaining steps as specified in “Enabling security for the realm” on page 84. As the final step, validate this setup by clicking **Apply** in the Global security panel.
3. Save, stop, and restart all the product servers (deployment managers, nodes, and Application Servers) for changes in this panel to take effect. If the server comes up without any problems, the setup is correct.

Member attributes collection:

Use this page to list Lightweight Directory Access Protocol (LDAP) member attributes or to select a member attribute to view or change its configuration properties.

To view this administrative console page, complete the following steps:

1. In the administrative console, click **Security > Global security**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**. To configure for a specific domain in a multiple security domain environment, click **Security domains > domain_name**. Under Security Attributes, expand User Realm, and click **Customize for this domain**. Select the Realm type as Federated repositories and then click **Configure**.
3. Under Related items, click **Manage repositories**.
4. Click **Add > LDAP repository** to specify a new external repository or select an external repository that is preconfigured.
5. Under Additional properties, click **Group attribute definition**.
6. Under Additional properties, click **Member attributes**.

When you finish adding or updating your federated repository configuration, go to the **Security > Global security** panel and click **Apply** to validate the changes.

Name:

Specifies the name of the member attribute in LDAP. For example, member and uniqueMember are two commonly used names of member attributes.

The member attribute is used to store the values that reference members that the group contains. For example, a group type with an object class groupOfNames has a member attribute named member; group

type with object class groupOfUniqueNames has a member attribute named uniqueMember. An LDAP repository supports multiple group types if multiple member attributes and their associated group object classes are specified.

Scope:

Specifies the scope of the member attribute.

Information	**Value**
Default:	Direct
Range:	**Direct** The member attribute contains direct members only. Direct members are members that are directly contained by the group. For example, if Group1 contains Group2 and Group2 contains User1, then User1 is a direct member of Group2, but User1 is not a direct member of Group1. **Nested** The member attribute contains both direct members and nested members. **All** The member attribute contains direct members, nested members, and dynamic members.

Object class:

Specifies the object class of the group that uses this member attribute. If this field is not defined, this member attribute applies to all group object classes.

Member attributes settings:

Use this page to configure Lightweight Directory Access Protocol (LDAP) member attributes.

To view this administrative console page, complete the following steps:

1. In the administrative console, click **Security > Global security**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**. To configure for a specific domain in a multiple security domain environment, click **Security domains > *domain_name***. Under Security Attributes, expand User Realm, and click **Customize for this domain**. Select the Realm type as **Federated repositories** and then click **Configure**.
3. Under Related items, click **Manage repositories**.
4. Click **Add > LDAP repository** to specify a new external repository or select an external repository that is pre-configured.
5. Under Additional properties, click **Group attribute definition**.
6. Under Additional properties, click **Member attributes**.
7. Click **New** to specify a new member attribute.

When you finish adding or updating your federated repository configuration, go to the **Security > Global security** panel and click **Apply** to validate the changes.

Name of member attribute:

Specifies the name of the member attribute in LDAP. For example, member and uniqueMember are two commonly used names of member attributes.

The member attribute is used to store the values that reference members that the group contains. For example, a group type with an object class groupOfNames has a member attribute named member; group type with object class groupOfUniqueNames has a member attribute named uniqueMember. An LDAP repository supports multiple group types if multiple member attributes and their associated group object classes are specified.

Object class:

Specifies the object class of the group that uses this member attribute. If this field is not defined, this member attribute applies to all group object classes.

Scope:

Specifies the scope of the member attribute.

Information	**Value**
Default:	Direct
Range:	**Direct** The member attribute contains direct members only. Direct members are members that are directly contained by the group. For example, if Group1 contains Group2 and Group2 contains User1, then User1 is a direct member of Group2, but User1 is not a direct member of Group1.
	Nested The member attribute contains both direct members and nested members.
	All The member attribute contains direct members, nested members, and dynamic members.

Configuring dynamic member attributes in a federated repository configuration

Follow this task to configure dynamic member attributes in a federated repository configuration.

Before you begin

Because dynamic member attributes apply only to a Lightweight Directory Access Protocol (LDAP) repository, you must first configure an LDAP repository. For more information, see “Managing repositories in a federated repository configuration” on page 300.

About this task

A dynamic group defines its members differently than a static group. Instead of listing the members individually, the dynamic group defines its members using an LDAP search. The filter for the search is defined in a dynamic member attribute. For example, the dynamic group uses the structural objectclass groupOfURLs, or auxiliary objectclass ibm-dynamicGroup, and the attribute memberURL, to define the search using a simplified LDAP URL syntax:

```
ldap:///<base DN of search> ? ? <scope of search> ? <searchfilter>
```

The following is an example of the LDAP URL that defines all entries that are under `o=Acme` with the `objectclass=person`:

```
ldap:///o=Acme,c=US??sub?objectclass=person
```

If both member and dynamic member attributes are specified for the same group type, this group type is a hybrid group with both static and dynamic members.

Procedure

1. In the administrative console, click **Security > Global security**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**. To configure for a specific domain in a multiple security domain environment, click **Security domains > *domain_name***. Under Security Attributes, expand User Realm, and click **Customize for this domain**. Select the Realm type as **Federated repositories** and then click **Configure**.
3. Under Related items, click **Manage repositories**.
4. Click **Add > LDAP repository** to specify a new external repository or select an external repository that is preconfigured.

   Note: If you click **Add** to specify a new external repository, you must first complete the required fields and click **Apply** before you can proceed to the next step.
5. Under **Additional Properties**, click **LDAP entity types**.
6. Click the link for **Group** entity type.
7. In the **Object Classes** field, add the entry for the object class, for example, groupOfUrls. Delimit multiple entries with a semicolon (;).
8. Click **OK**.
9. Under Additional properties, click **Group attribute definition**.
10. Under Additional properties, click **Dynamic member attributes**.
11. Click **New** to specify a new dynamic member attribute or **Delete** to remove a preconfigured dynamic member attribute.
12. Specify the name of the dynamic member attribute in the Name of dynamic member attribute field. The name of the dynamic member attribute defines the filter for dynamic group members in LDAP, for example, memberURL is the name of a commonly used dynamic member attribute.
13. Specify the object class of the group that contains the dynamic member attribute in the Dynamic object class field, for example, groupOfURLs. If this property is not defined, the dynamic member attribute applies to all group object classes.
14. Save your configuration changes in the administration console: Click **System administration > Save changes to master repository > Save**.
15. This next step involves using a wsadmin command and cannot be done through the administrative console. Start the wsadmin scripting tool and connect to a server, by using the following command:

    ```
    wsadmin –username username –password password
    ```
16. Use the **addIdMgrPropertyToEntityTypes** command to add the dynamic member attribute specified in step 12 to the federated repositories schema. The dynamic member attribute needs to be added to the entity type Group in the federated repositories schema otherwise an error occurs while creating a group in an LDAP repository configured under federated repositories using the create() API and specifying the memberURL attribute and its value. The correctness of the value of the memberURL attribute is not validated because LDAP does not validate this.

    In the following example, the memberURL property is added to the entity type Group:

    ```
    $AdminTask addIdMgrPropertyToEntityTypes {-name memberURL -dataType String -entityTypeNames Group -repositoryIds repos
    ```
17. Save your configuration changes.

    ```
    $AdminConfig save
    ```

Results

After completing these steps, dynamic member attributes are configured for your LDAP repository.

What to do next

1. After configuring the federated repositories, click **Security > Global security** to return to the Global security panel. Verify that Federated repositories is identified in the Current realm definition field. If

Federated repositories is not identified, select **Federated repositories** from the Available realm definitions field and click **Set as current**. To verify the federated repositories configuration, click **Apply** on the Global security panel. If Federated repositories is not identified in the Current realm definition field, your federated repositories configuration is not used by WebSphere Application Server.

2. If you are enabling security, complete the remaining steps as specified in "Enabling security for the realm" on page 84. As the final step, validate this setup by clicking **Apply** in the Global security panel.
3. Save, stop, and restart all the product servers (deployment managers, nodes, and Application Servers) for changes in this panel to take effect. If the server comes up without any problems, the setup is correct.

Dynamic member attributes collection:

Use this page to manage Lightweight Directory Access Protocol (LDAP) dynamic member attributes.

To view this administrative console page, complete the following steps:

1. In the administrative console, click **Security > Global security**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**. To configure for a specific domain in a multiple security domain environment, click **Security domains > *domain_name***. Under Security Attributes, expand User Realm, and click **Customize for this domain**. Select the Realm type as **Federated repositories** and then click **Configure**.
3. Under Related items, click **Manage repositories**.
4. Click **Add > LDAP repository** to specify a new external repository or select an external repository that is preconfigured.
5. Under Additional properties, click **Group attribute definition**.
6. Under Additional properties, click **Dynamic member attributes**.

When you finish adding or updating your federated repository configuration, go to the **Security > Global security** panel and click **Apply** to validate the changes.

Name:

Specifies the name of the attribute that defines the filter for dynamic group members in LDAP. For example, memberURL is the name of a commonly used dynamic member attribute.

If both member and dynamic member attributes are specified for the same group type, this group type is a hybrid group with both static and dynamic members.

A dynamic group defines its members differently than a static group. Instead of listing the members individually, the dynamic group defines its members using an LDAP search. The filter for the search is defined in a dynamic member attribute. For example, the dynamic group uses the structural objectclass groupOfURLs, or auxiliary objectclass ibm-dynamicGroup, and the attribute memberURL, to define the search using a simplified LDAP URL syntax:

```
ldap:///<base DN of search>??<scope of search>?<searchfilter>
```

The following is an example of the LDAP URL that defines all entries that are under `o=Acme` with the `objectclass=person`:

```
ldap:///o=Acme,c=US??sub?objectclass=person
```

Object class:

Specifies the object class of the group that contains this dynamic member attribute, for example, groupOfURLs. If this property is not defined, the dynamic member attribute applies to all group object classes.

Dynamic member attributes settings:

Use this page to configure Lightweight Directory Access Protocol (LDAP) dynamic member attributes.

To view this administrative console page, complete the following steps:

1. In the administrative console, click **Security > Global security**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**. To configure for a specific domain in a multiple security domain environment, click **Security domains > *domain_name***. Under Security Attributes, expand User Realm, and click **Customize for this domain**. Select the Realm type as **Federated repositories** and then click **Configure**.
3. Under Related items, click **Manage repositories**.
4. Click **Add > LDAP repository** to specify a new external repository or select an external repository that is preconfigured.
5. Under Additional properties, click **Group attribute definition**.
6. Under Additional properties, click **Dynamic member attributes**.
7. Click **New** to specify a new dynamic member attribute.

When you finish adding or updating your federated repository configuration, go to the **Security > Global security** panel and click **Apply** to validate the changes.

Name of dynamic member attribute:

Specifies the name of the attribute that defines the filter for dynamic group members in LDAP. For example, memberURL is the name of a commonly used dynamic member attribute.

If both member and dynamic member attributes are specified for the same group type, this group type is a hybrid group with both static and dynamic members.

A dynamic group defines its members differently than a static group. Instead of listing the members individually, the dynamic group defines its members using an LDAP search. The filter for the search is defined in a dynamic member attribute. For example, the dynamic group uses the structural objectclass groupOfURLs, or auxiliary objectclass ibm-dynamicGroup, and the attribute memberURL, to define the search using a simplified LDAP URL syntax:

```
ldap:///<base DN of search>??<scope of search>?<searchfilter>
```

The following is an example of the LDAP URL that defines all entries that are under `o=Acme` with the `objectclass=person`:

```
ldap:///o=Acme,c=US??sub?objectclass=person
```

Dynamic object class:

Specifies the object class of the group that contains this dynamic member attribute, for example, groupOfURLs. If this property is not defined, the dynamic member attribute applies to all group object classes.

Standalone Lightweight Directory Access Protocol registries

A Standalone Lightweight Directory Access Protocol (LDAP) registry performs authentication using an LDAP binding.

WebSphere Application Server security provides and supports the implementation of most major LDAP directory servers, which can act as the repository for user and group information. These LDAP servers are called by the product processes for authenticating a user and other security-related tasks. For example, the servers are used to retrieve user or group information. This support is provided by using different user and group filters to obtain the user and group information. These filters have default values that you can modify to fit your needs. The custom LDAP feature enables you to use any other LDAP server, which is not in the product-supported list of LDAP servers, for its user registry by using the appropriate filters.

Note: The initial profile creation configures WebSphere Application Server to use a federated repositories security registry option with the file-based registry. This security registry configuration can be changed to use other options, including the stand-alone LDAP registry. Instead of changing from the federated repositories option to the stand-alone LDAP registry option under the User account repository configuration, consider employing the federated repositories option, which provides for LDAP configuration. Federated repositories provide a wide range of capabilities, including the ability to have one or multiple user registries. It supports federating one or more LDAPs in addition to file-based and custom registries. It also has improved failover capabilities, and a robust set of member (user and group) management capabilities. Federated repositories is required when you are using the new member management capabilities in WebSphere Portal 6.1 and higher, and Process Server 6.1 and higher. The use of federated repositories is required for following LDAP referrals, which is a common requirement in some LDAP server environments (such as Microsoft Active Directory).

It is recommended that you migrate from stand-alone LDAP registries to federated repositories. If you move to WebSphere Portal 6.1 and higher, and or WebSphere Process Server 6.1 and higher, you should migrate to federated repositories prior to these upgrades. For more information about federated repositories and its capabilities, read the Federated repositories topic. For more information about how to migrate to federated repositories, read the Migrating a stand-alone LDAP repository to a federated repositories LDAP repository configuration topic.

To use LDAP as the user registry, you need to know an administrative user name that is defined in the registry, the server host and port, the base distinguished name (DN) and, if necessary, the bind DN and the bind password. You can choose any valid user in the registry that is searchable and have administrative privileges. In some LDAP servers, the administrative users are not searchable and cannot be used, for example, `cn=root` in SecureWay. This user is referred to as WebSphere Application Server security server ID, server ID, or server user ID in the documentation. Being a server ID means a user has special privileges when calling some protected internal methods. Normally, this ID and password are used to log into the administrative console after security is turned on. You can use other users to log in if those users are part of the administrative roles.

When security is enabled in the product, the primary administrative user name and password are authenticated with the registry during the product startup. If authentication fails, the server does not start. It is important to choose an ID and password that do not expire or change often. If the product server user ID or password need to change in the registry, make sure that the changes are performed when all the product servers are up and running.

When the changes are done in the registry, use the steps that are described in “Configuring Lightweight Directory Access Protocol user registries” on page 170. Change the ID, password, and other configuration information, save, stop, and restart all the servers so that the new ID or password is used by the product. If any problems occur starting the product when security is enabled, disable security before the server can

start up. To avoid these problems, make sure that any changes in this panel are validated in the Global security panel. When the server is up, you can change the ID, password, and other configuration information and then enable security.

You can use the custom Lightweight Directory Access Protocol (LDAP) feature to support any LDAP server by setting up the correct configuration. However, support is not extended to these custom LDAP servers because many configuration possibilities exist.

The users and groups and security role mapping information is used by the configured authorization engine to perform access control decisions.

Dynamic groups and nested group support for LDAP

Dynamic and nested groups simplify WebSphere Application Server security management and increase its effectiveness and flexibility.

Dynamic groups contain a group name and membership criteria:

- The group membership information is as current as the information on the user object.
- There is no need to manually maintain members on the group object.
- Dynamic groups are designed so an application does not need a large amount of information from the directory to find out if someone is a member of a group.

Nested groups enable the creation of hierarchical relationships that are used to define inherited group membership. A nested group is defined as a child group entry whose distinguished name (DN) is referenced by a parent group entry attribute.

You only need to assign a larger parent group if all nested groups share the same privilege. Assigning a role to a single parent group simplifies the run-time authorization table.

Dynamic groups and nested group support for the IBM Tivoli Directory Server

WebSphere Application Server supports all Lightweight Directory Access Protocol (LDAP) dynamic and nested groups when using IBM Tivoli Directory Server. This function is enabled by default by taking advantage of a new feature in IBM Tivoli Directory Server. IBM Tivoli Directory Server uses the ibm-allGroups forward-reference group attribute that automatically calculates all the group memberships including dynamic and recursive memberships for a user. Security directly locates a user group membership from a user object rather than indirectly search all the groups to match group members.

For more information, see “Configuring dynamic and nested group support for the IBM Tivoli Directory Server” on page 191.

Dynamic and nested group support for the SunONE or iPlanet Directory Server

The SunONE or iPlanet Directory Server uses two grouping mechanisms:

Groups
Entries that name other entries as a list of members or as a filter for members.

Roles Entries that name other entries as a list of members or as a filter for members. Additional functionality is provided by generating the nsrole attribute on each role member.

Three types of roles are available:

Filtered roles
Depends upon the attributes that are contained in each entry. Entries are members, if they match a specified Lightweight Directory Access Protocol (LDAP) filter. This role is equivalent to a dynamic group.

Nested roles
Creates roles that contain other roles. This role is equivalent to a nested group.

Managed roles
Explicitly assigns a role to member entries. This role is equivalent to a static group.

Refer to “Configuring dynamic and nested group support for the SunONE or iPlanet Directory Server” on page 191 for more information.

Security failover among multiple LDAP servers

WebSphere Application Server security can be configured to attempt failovers between multiple Lightweight Directory Access Protocol (LDAP) hosts.

Note: This topic references one or more of the application server log files. As a recommended alternative, you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log` , `SystemErr.log`, `trace.log`, and `activity.log` files on distributed and IBM i systems. You can also use HPEL in conjunction with your native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

If the current active LDAP server is unavailable, WebSphere Application Server security attempts a failover to the first available LDAP host in the specified host list. The multiple LDAP servers can be replicas of the same master LDAP server, or they can be any LDAP host with the same schema, which contain data that is imported from the same LDAP Data Interchange Format (LDIF) file.

Whenever a failover occurs, WebSphere Application Server security always uses the first available LDAP server in the specified host list. For example, if there are four LDAP servers configured in the order of L1, L2, L3, and L4, L1 is treated as the primary LDAP server. The preference of connection is from L1 to L4. If, for example, WebSphere Application Server security is currently connected to L4, and failover or reconnection is necessary, WebSphere Application Server security first attempts to connect to L1, L2, and then L3 in that order until the connection is successful.

The current LDAP host name is logged in message SECJ0419I in the WebSphere Application Server log file, `SystemOut.log`. If you want to reconnect to the primary LDAP host, run the WebSphere Application Server MBean method, resetLDAPBindInfo, with `null,null` as the input.

To configure LDAP failover among multiple LDAP hosts, you must use wsadmin or ConfigService to include the backup LDAP host, which does not have a number limitation. The LDAP host that is displayed in the administrative console is the primary LDAP host, and is the first item listed in the LDAP host list in `security.xml`.

The WebSphere Application Server security realm name defaults to the primary LDAP host name that is displayed in the administrative console. It includes a trailing colon and a port number (if one exists). However, the custom property, com.ibm.websphere.security.ldap.logicRealm, can be added to override the default security realm name. Use the logicRealm name to configure each cell to have its own LDAP host for interoperability and backward compatibility, and to provide flexibility for adding or removing the LDAP host dynamically. If migrating from a previous installation, the new logicRealm name does not take effect until administrative security is enabled again. To be compatible with a previous release that does not support logic realm, the logicRealm name has to be the same as that used by the previous installation (the LDAP host name, including a trailing colon and port number).

When LDAP failover is configured by associating a single hostname to multiple IP addresses through the use of a load balancer (which does that translation transparently to WebSphere Application Server), entering an invalid password can cause multiple LDAP bind retries. WebSphere Application Server retries and the load balancer routes requests to multiple replicas. With the default settings, the number of LDAP bind retries is equal to one more than the number of associated IP addresses. This means a single invalid

login attempt can cause the LDAP account to be locked. If the com.ibm.websphere.security.registry.ldap.singleLDAP custom property is set to `false`, LDAP bind calls are not retried.

When LDAP failover is configured by registering backend LDAP server hostnames using wsadmin command, set the com.ibm.websphere.security.ldap.retryBind property to false.

The following Jacl example shows how to use wsadmin to add a backup LDAP host for failover:

```
#-------------------------------------------------------------------
# Main
#  This is a bi-modal script: it can be included in the wsadmin
#  command invocation like this:
#     wsadmin -f LDAPAdd.jacl ldaphost 800
#
#  or the script can be sourced from the wsadmin command line:
#     wsadmin> source LDAPAdd.jacl
#     wsadmin> LDAPAdd ldaphost 800
#
#  The script expects some parameters:
#      arg1 - LDAP Server host name
#      arg2 - LDAP Server port number
#-------------------------------------------------------------------
if { !($argc == 2)} {
   puts ""
   puts "LDAPAdd: This script requires 2 parameters: LDAP server host name and LDAP server port number"
   puts "For example: LDAPAdd ldaphost 389"
   puts ""
   return;
}
else {
   set ldapServer        [lindex $argv 0]
   set ldapPort          [lindex $argv 1]
   LDAPAdd $ldapServer $ldapPort
   return;
}
proc LDAPAdd {ldapServer ldapPort args} {
   global AdminConfig AdminControl ldapServer ldapPort
   set ldapServer  lindex $args 0
   set ldapPort  lindex $args 1
   global ldapUserRegistryId
   # Get the LDAP user registry object from the security configuration
   if { catch {$AdminConfig list LDAPUserRegistry} result } {
      puts stdout "\$AdminConfig list LDAPUserRegistry caught an exception $result\n"
      return
   }
   else {
      if {$result != {}} {
         set ldapUserRegistryId  lindex $result 0
      }
      else {
         puts stdout "\$AdminConfig list LDAPUserRegistry caught an exception $result\n"
         return;
      }
   }
   # Set the host and port values in Attrs2
   set Attrs2  list  list hosts  list  list  list host
   $ldapServer
   list port $ldapPort

   # Modify the LDAP configuration host object
   $AdminConfig modify $ldapUserRegistryId $Attrs2
   $AdminConfig save
}
```

The following Jython example shows how to use wsadmin to add a backup LDAP host for failover:

```
#-------------------------------------------------------------------
# Add ldap hostname and port
#     wsadmin -f LDAPAdd.py arg1 arg2
#
#  The script expects some parameters:
#      arg1 - LDAP Server hostname
#      arg2 - LDAP Server portnumber
#-------------------------------------------------------------------
import java

#---------------------------------------------------------
# get the line separator and use to do the parsing
# since the line separator on different platform are different
lineSeparator = java.lang.System.getProperty('line.separator')

#--------------------------------------------------------------------------------
# add LDAP host
#--------------------------------------------------------------------------------
```

```
def LDAPAdd (ldapServer, ldapPort):
    global AdminConfig, lineSeparator, ldapUserRegistryId
    try:
        ldapObject = AdminConfig.list("LDAPUserRegistry")
        if len(ldapObject) == 0:
            print "LDAPUserRegistry ConfigId was not found\n"
            return

        ldapUserRegistryId = ldapObject.split(lineSeparator)[0]
        print "Got LDAPUserRegistry ConfigId is " + ldapUserRegistryId + "\n"
    except:
        print "AdminConfig.list('LDAPUserRegistry') caught an exception\n"

    try:
        secMbeans = AdminControl.queryNames('WebSphere:type=SecurityAdmin,*')
        if len(secMbeans) == 0:
            print "Security Mbean was not found\n"
            return

        secMbean = secMbeans.split(lineSeparator)[0]
        print "Got Security Mbean is " + secMbean + "\n"
    except:
        print "AdminControl.queryNames('WebSphere:type=SecurityAdmin,*') caught an exception\n"


    attrs2 = [["hosts", [[["host", ldapServer], ["port", ldapPort]]]]]
    try:
        AdminConfig.modify(ldapUserRegistryId, attrs2)
        try:
            AdminConfig.save()
            print "Done setting up attributes values for LDAP User Registry"
            print "Updated was saved successfully\n"
        except:
            print "AdminConfig.save() caught an exception\n"
    except:
        print "AdminConfig.modify(" + ldapUserRegistryId + ", " + attrs2 + ") caught an exception\n"
    return

#-------------------------------------------------------------------------------
# Main entry point
#-------------------------------------------------------------------------------
if len(sys.argv) < 2 or len(sys.argv) > 3:
        print("LDAPAdd: this script requires 2 parameters: LDAP server hostname and LDAP server port number\n")
        print("e.g.: LDAPAdd ldaphost 389\n")
        sys.exit(1)
else:
        ldapServer = sys.argv[0]
        ldapPort = sys.argv[1]
        LDAPAdd(ldapServer, ldapPort)
        sys.exit(0)
```

Selecting an authentication mechanism

An *authentication mechanism* defines rules about security information, such as whether a credential is forwardable to another Java process, and the format of how security information is stored in both credentials and tokens. You can select and configure an authentication mechanism by using the administrative console.

About this task

Authentication is the process of establishing whether a client is who or what it claims to be in a particular context. A client can be either an end user, a machine, or an application. An authentication mechanism in WebSphere Application Server typically collaborates closely with a *user registry*. The user registry is the user and groups account repository that the authentication mechanism consults with when performing authentication. The authentication mechanism is responsible for creating a *credential*, which is an internal product representation of a successfully authenticated client user. Not all credentials are created equally. The abilities of the credential are determined by the configured authentication mechanism.

WebSphere Application Server provides three authentication mechanisms: Lightweight Third Party Authentication (LTPA), Kerberos, and RSA token authentication mechanism.

Security support for Kerberos as the authentication mechanism has been added for this release of WebSphere Application Server. Kerberos (KRB5) is a mature, flexible, open, and very secure network authentication protocol. Kerberos includes authentication, mutual authentication, message integrity and

confidentiality and delegation features. KRB5 is used for Kerberos in the administrative console and in the **sas.client.props**, **soap.client.props** and **ipc.client.props** files.

The RSA token authentication mechanism is new to this release of WebSphere Application Server. It aids the flexible management objective to preserve the base profiles configurations and isolate them from a security perspective. This mechanism permits the base profiles managed by an administrative agent to have different Lightweight Third-Party Authentication (LTPA) keys, different user registries, and different administrative users.

Note: Simple WebSphere Authentication Mechanism (SWAM) is deprecated in this release. SWAM does not provide authenticated communication between different servers.

Authentication is required for enterprise bean clients and web clients when they access protected resources. Enterprise bean clients, like a servlet or other enterprise beans or a pure client, send the authentication information to a web application server using one of the following protocols:

- Common Secure Interoperability Version 2 (CSIv2)
- Secure Authentication Service (SAS)

Note: SAS is supported only between Version 6.0.x and previous version servers that have been federated in a Version 6.1 cell.

Web clients use the HTTP or HTTPS protocol to send the authentication information.

The authentication information can be basic authentication (user ID and password), a credential token, or a client certificate. The web authentication is performed by the web authentication module.

You can configure web authentication for a web client by using the administrative console. Click **Security > Global security**. Under Authentication, expand **Web and SIP security** and click **General settings**. The following options exist for Web authentication:

Authenticate only when the URI is protected
Specifies that the web client can retrieve an authenticated identity only when it accesses a protected Uniform Resource Identifier (URI). WebSphere Application Server challenges the web client to provide authentication data when the web client accesses a URI that is protected by a J2EE role. This default option is also available in previous versions of WebSphere Application Server.

Use available authentication data when an unprotected URI is accessed
Specifies that the web client is authorized to call the `getRemoteUser`, `isUserInRole`, and `getUserPrincipal` methods; retrieves an authenticated identity from either a protected or an unprotected URI. Although the authentication data is not used when you access an unprotected URI, the authentication data is retained for future use. This option is available when you select the **Authentication only when the URI is protected** check box.

Authenticate when any URI is accessed
Specifies that the web client must provide authentication data regardless of whether the URI is protected.

Default to basic authentication when certificate authentication for the HTTPS client fails.
Specifies that WebSphere Application Server challenges the web client for a user ID and password when the required HTTPS client certificate authentication fails.

The enterprise bean authentication is performed by the Enterprise JavaBeans (EJB) authentication module.

The EJB authentication module resides in the CSIv2 and SAS layer.

The authentication module is implemented using the Java Authentication and Authorization Service (JAAS) login module. The web authenticator and the EJB authenticator pass the authentication data to the login module, which can use the following mechanisms to authenticate the data:

- Kerberos
- LTPA
- RSA token
- Simple WebSphere Authentication Mechanism (SWAM)

Note: SWAM was deprecated in WebSphere Application Server Version 8.5 and will be removed in a future release.

The authentication module uses the registry that is configured on the system to perform the authentication. Four types of registries are supported:

- Federated repositories
- Local operating system
- Standalone Lightweight Directory Access Protocol (LDAP) registry
- Stand-alone custom registry

External registry implementation following the registry interface that is specified by IBM can replace either the local operating system or the LDAP registry.

The login module creates a JAAS subject after authentication and stores the credential that is derived from the authentication data in the public credentials list of the subject. The credential is returned to the web authenticator or to the enterprise beans authenticator.

The web authenticator and the enterprise beans authenticator store the received credentials in the Object Request Broker (ORB) current for the authorization service to use in performing further access control checks. If the credentials are forwardable, they are sent to other application servers.

You can configure authentication mechanisms in the administrative console by doing the following:

Procedure

1. **Click Security > Global security.**
2. Under Authentication mechanisms and expiration, select an authentication mechanism to configure.

Lightweight Third Party Authentication

Lightweight Third Party Authentication (LTPA) is intended for distributed, multiple application server and machine environments. LTPA supports forwardable credentials and single sign-on (SSO). LTPA can support security in a distributed environment through cryptography. This support permits LTPA to encrypt, digitally sign, and securely transmit authentication-related data, and later decrypt and verify the signature.

Application servers can securely communicate using the LTPA protocol. It also provides the single sign-on (SSO) feature wherein a user is required to authenticate only once in a domain name system (DNS) domain and can access resources in other WebSphere Application Server cells without getting prompted. The realm names on each system in the DNS domain are case sensitive and must match identically.

AIX HP-UX Solaris For local OS, the realm name is the same as the host name.

Windows For local OS, the realm name is the domain name, if a domain is in use or the realm name is the machine name.

For the Lightweight Directory Access Protocol (LDAP), the realm name is the `host:port` value of the LDAP server.

The LTPA protocol uses cryptographic keys to encrypt and decrypt user data that passes between the servers. These keys must be shared between the different cells for the resources in one cell to access resources in other cells, assuming that all the cells involved use the same LDAP or custom registry.

When using LTPA, a token is created with the user information and an expiration time and is signed by the keys. The LTPA token is time sensitive. All product servers that participate in a protection domain must have their time and date synchronized. If not, LTPA tokens appear prematurely expired and cause authentication or validation failures. Coordinated Universal Time (UTC) is used by default, and all other machines must have the same UTC time. Consult your operating system documentation for information regarding how to ensure this.

This token passes to other servers, in the same cell or in a different cell through cookies, for web resources when SSO is enabled, or through the authentication protocol layer for enterprise beans.

If the receiving servers share the same keys as the originating server, the token can be decrypted to obtain the user information, which then is validated to make sure that it has not expired and that the user information in the token is valid in its registry. On successful validation, the resources in the receiving servers are accessible after the authorization check.

Each server must have valid credentials. When the credentials expire, the server is required to communicate to the user registry to authenticate. User registry outages can cause server processes to hang, requiring them to be restarted to recover. Extending the time the LTPA token remains cached reduces this risk, but does present a slightly increased security risk to be considered when defining your security policies.

All of the WebSphere Application Server processes in a cell share the same set of keys. If key sharing is required between different cells, export them from one cell and import them to the other. For security purposes, the exported keys are encrypted with a random generated key and a user-defined password is used to protect the keys. This same password is needed when importing the keys into another cell. The password is only used to protect the keys and is not used to generate the keys.

WebSphere Application Server supports the LTPA, Kerberos and the Simple WebSphere Authentication Mechanism (SWAM) protocols.

Note: SWAM is deprecated in WebSphere Application Server Version 8.5 and will be removed in a future release.

When security is enabled during profile creation time, LTPA is configured by default.

LTPA requires that the configured user registry be a centrally shared repository such as LDAP or a Windows domain-type registry so that users and groups are the same, regardless of the machine.

Lightweight Third Party Authentication key sets and key set groups

Key set groups contain lists of key sets and Lightweight Third Party Authentication (LTPA) key generation schedules. Each key set contains key references to keys in key stores.

Note: It is not recommended that you choose to generate new keys automatically . Keys should only be generated during off hours. Once keys are generated, you might need to export the keys and to import the keys to other WebSphere cells or IBM products in which the keys are required to be sync to communicate with each other.

The keys for some key configurations must be generated together. The LTPA key pair is referenced in one key set while the secret or private key is in a separate key set. When the key set group is created, the two key sets are added as members of the key set group. Key set group settings determine whether the keys for both key sets are generated together automatically or manually.

The key set group contains the following attributes:

- Member key sets
- Choice of either manual or automatic key generation in the member key sets
- Schedule for automatically generating keys

Configuring LTPA and working with keys

You must configure Lightweight Third Party Authentication (LTPA) when you set up security for the first time. LTPA is the default authentication mechanism for WebSphere Application Server. After you have configured LTPA you can generate LTPA keys manually or automatically.

Procedure

1. Configure LTPA and generate the first LTPA keys. Use the administrative console to configure LTPA or Kerberos when you set up security for the first time. The LTPA keys are generated automatically the first time. Read the Configuring the Lightweight Third Party Authentication mechanism article for more information.

   Application servers distributed in multiple nodes and cells can securely communicate using the LTPA protocol. Key set groups contain lists of key sets and LTPA authentication key generation schedules. Each key set contains key references to keys in key stores. To generate keys automatically, each key set must be a member of a key set group.

   Read the Lightweight Third Party Authentication key sets and key set groups article for more information.

   The keys for some key configurations must be generated together. The LTPA key pair is referenced in one key set while the secret or private key is in a separate key set. When the key set group is created, the two key sets are added as members of the key set group. Key set group settings determine whether the keys for both key sets are generated together automatically or manually.

   The key set group contains the following attributes:

   - Member key sets
   - Choice of either manual or automatic key generation in the member key sets
   - Schedule for automatically generating keys

2. Generate keys manually or automatically, and control the number of active keys. WebSphere Application Server generates Lightweight Third Party Authentication (LTPA) keys automatically during the first server startup. You can generate additional keys as you need them in the Authentication mechanisms and expiration panel.

   You can disable the automatic generation of new LTPA keys for key sets that are members of a key set group. Automatic generation creates new keys on a schedule that you specify when you configure a key set group, which manages one or more key sets. WebSphere Application Server uses key set groups to automatically generate cryptographic keys or multiple synchronized key sets.

   Generating keys manually or enabling or disabling the generation of keys are tasks that require you to recycle the node agents and application servers to accept the new keys. If any of the node agents are down, run a manual file synchronization utility from the node agent machine to synchronize the security configuration from the deployment manager.

   Key sets manage LTPA keys in a key store that is based on a key alias prefix. A key alias prefix is automatically generated when you generate a new key and store it in a key store. Key stores can contain multiple versions of keys for any given key alias prefix. You can specify a maximum number of active keys in the key set configuration.

   Read the Generating Lightweight Third Party Authentication keys article for more information.

3. Import and export keys. To support single sign-on (SSO) in WebSphere® Application Server across multiple WebSphere Application Server domains or cells, you must share the LTPA keys and the password among the domains. You can import LTPA keys from other domains and export keys to other domains.

Note: You should disable automatic key generation if you import or export keys to or from another cell. This disabling causes the imported keys to get lost and the exported keys to no longer interoperate with this cell over time

You must recycle the node agents and application servers to accept the new keys. If any of the node agents are down, run a manual file synchronization utility from the node agent machine to synchronize the security configuration from the deployment manager.

Read the Importing Lightweight Third Party Authentication keys and Exporting Lightweight Third Party Authentication keys articles for more information.

4. Manage keys from multiple cells. You can specify the shared keys and configure the authentication mechanism that is used to exchange information between servers to import and export LTPA keys across multiple WebSphere® Application Server cells.

   You must start the server again for any changes you make to become active.

   Read the Managing LTPA keys from multiple WebSphere Application Server cells article for more information.

Kerberos (KRB5) authentication mechanism support for security

The Kerberos authentication mechanism enables interoperability with other applications (such as .NET, DB2 and others) that support Kerberos authentication. It provides single sign on (SSO) end-to-end interoperable solutions and preserves the original requester identity.

Note: Security support for Kerberos as the authentication mechanism was added for WebSphere Application Server Version 7.0. Kerberos is a mature, flexible, open, and very secure network authentication protocol. Kerberos includes authentication, mutual authentication, message integrity and confidentiality and delegation features. You can enable Kerberos on the server side. Support is provided to enable the rich Java client to use the Kerberos token for authentication to the WebSphere Application Server.

The following sections describe Kerberos authentication in more detail:

What is Kerberos?

Kerberos has withstood the test of time and is now at version 5.0. Kerberos enjoys wide spread platform support (for example, for Windows, Linux, Solaris, AIX, and z/OS) partly because the Kerberos source code is freely downloadable from the Massachusetts Institute of Technology (MIT) where it was originally created.

Kerberos is composed of three parts: a client, a server, and a trusted third party known as the Kerberos Key Distribution Center (KDC). The KDC provides authentication and ticket granting services.

The KDC maintains a database or repository of user accounts for all of the security principals in its realm. Many Kerberos distributions use file-based repositories for the Kerberos principal and policy DB and others use Lightweight Directory Access Protocol (LDAP) as the repository.

Kerberos does not support any notion of groups (that is, iKeys groups or groups of users or principals). The KDC maintains a long-term key for each principal in its accounts database. This long-term key is derived from the password of the principal. Only the KDC and the user that the principal represents should know what the long-term key or password is.

The benefits of having Kerberos as an authentication mechanism

The benefits of having Kerberos as the authentication mechanism for WebSphere Application Server include the following:

- The Kerberos protocol is a standard. This enables interoperability with other applications (such as .NET, DB2 and others) that support Kerberos authentication. It provides single sign on (SSO) end-to-end interoperable solutions and preserves the original requester identity.
- When using Kerberos authentication, the user clear text password never leaves the user machine. The user authenticates and obtains a Kerberos ticket granting ticket (TGT) from a KDC by using a one-way hash value of the user password. The user also obtains a Kerberos service ticket from the KDC by using the TGT. The Kerberos service ticket that represents the client identity is sent to WebSphere Application Server for authentication.
- A Java client can participate in Kerberos SSO using the Kerberos credential cache to authenticate to WebSphere Application Server.
- J2EE, web service, .NET and web browser clients that use the HTTP protocol can use the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) token to authenticate to the WebSphere Application Server and participate in SSO by using SPNEGO web authentication. Support for SPNEGO as the web authentication service is new to this release of WebSphere Application Server.

  Read about “Single sign-on for HTTP requests using SPNEGO web authentication” on page 376 for more information.
- WebSphere Application Server can support both Kerberos and Lightweight Third-Party Authentication (LTPA) authentication mechanisms at the same time.
- Server-to-server communication using Kerberos authentication is provided.

Kerberos authentication in a single Kerberos realm environment

WebSphere Application Server supports Kerberos authentication in a single Kerberos realm environment as shown in the following figure:

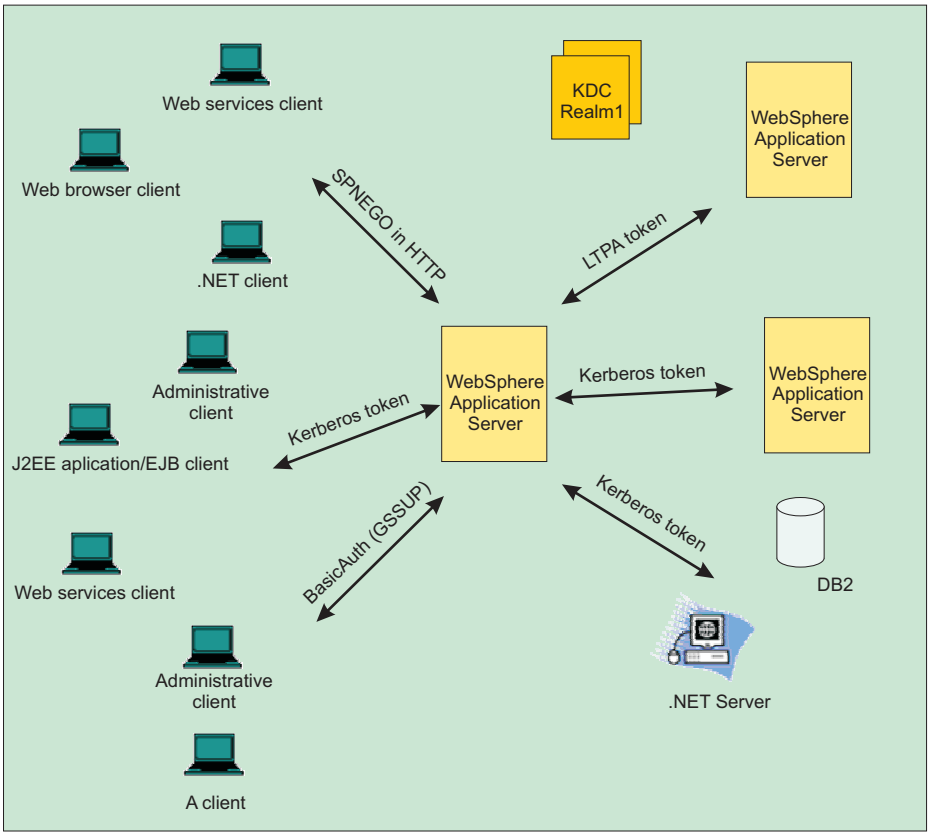


Figure 7. Kerberos authentication in a single Kerberos realm environment

When the WebSphere Application Server receives a Kerberos or SPNEGO token for authentication, it uses the Kerberos service principal (SPN) to establish a security context with a requestor. If a security context is established, the WebSphere Kerberos login module extracts a client GSS delegation credential, creates a Kerberos authentication token base on the Kerberos credential, and places them in the client subject with other tokens.

If the server must use a downstream server or back-end resources, it uses the client GSS delegation credential. If a downstream server does not support Kerberos authentication, the server uses the LTPA token instead of the Kerberos token. If a client does not include a GSS delegation credential in the request, the server uses the LTPA token for the downstream server . The Kerberos authentication token and principal are propagated to the downstream server as part of the security attributes propagation feature.

If the WebSphere Application Server and the KDC do not use the same user registry, then a JAAS custom login module might be required to map the Kerberos principal name to the WebSphere user name.

Kerberos authentication in a cross or trusted Kerberos realm environment

WebSphere Application Server also supports Kerberos authentication in a cross or trusted Kerberos realm environment as shown in the following figure:

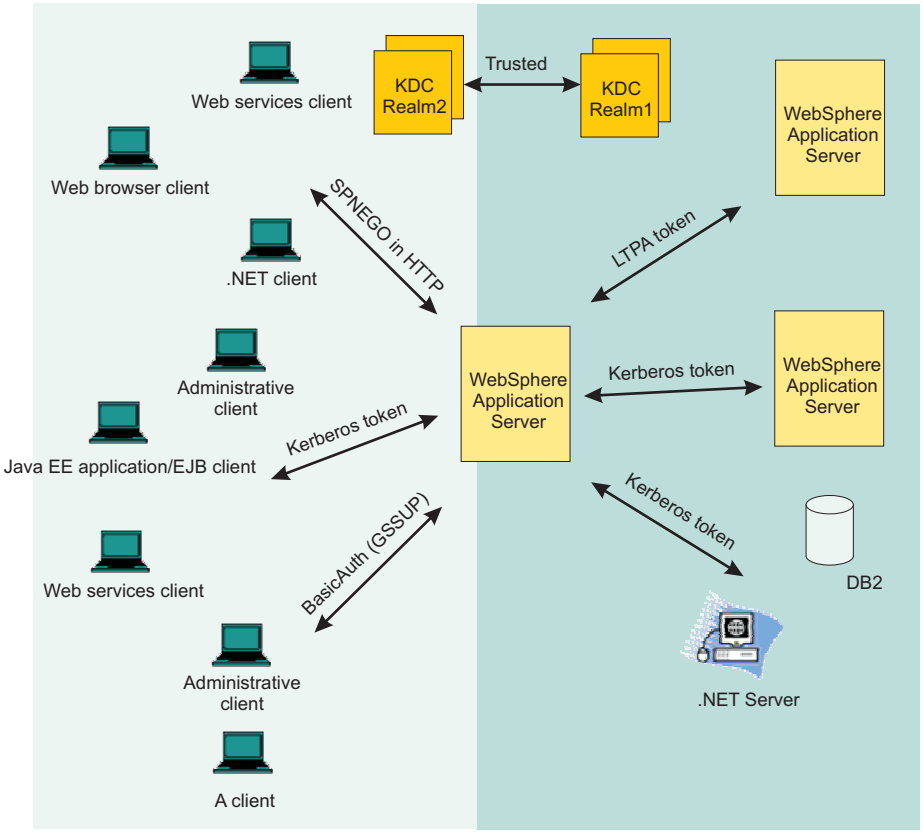


Figure 8. Kerberos authentication in a cross or trusted Kerberos realm environment

When the WebSphere Application Server receives a Kerberos or SPNEGO token for authentication, it uses the Kerberos service principal (SPN) to establish a security context with a requestor. If a security context is established, the WebSphere Kerberos login module always extracts a client GSS delegation credential and Kerberos ticket and places them in the client subject with other tokens.

If the server must use a downstream server or backend resources, it uses the client GSS delegation credential. If a downstream server does not support Kerberos authentication, the server uses the LTPA token instead of the Kerberos token. If a client does not include a GSS delegation credential in the request, the server uses the LTPA token for the downstream server . The Kerberos authentication token and principal are propagated to the downstream server as part of the security attributes propagation feature.

If the WebSphere Application Server and the KDC do not use the same user registry, then a JAAS custom login module might be required to map the Kerberos principal name to the WebSphere user name.

In this release of WebSphere Application Server, the new security multiple domains only support Kerberos at the cell level. All WebSphere Application Servers must be used by the same Kerberos realm. However, the clients and or backend resources (such as DB2, .NET server, and others) that support Kerberos authentication can have their own Kerberos realm. Only peer-to-peer and transitive trust cross-realm authentication are supported. The following steps must be performed for trusted Kerberos realms:

- The Kerberos trusted realm setup must be done on each of the Kerberos KDCs. See your Kerberos Administrator and User's guide for more information about how to set up a Kerberos trusted realm.
- The Kerberos configuration file might need to list the trusted realm.
- Add Kerberos trusted realms in the administrative console by clicking **Global security** > **CSIv2 outbound communications** > **Trusted authentication realms - outbound**.

The following figure shows a Java and administrative client that uses a Kerberos credential cache to authenticate to WebSphere Application Server with a Kerberos token in a trusted Kerberos realm:

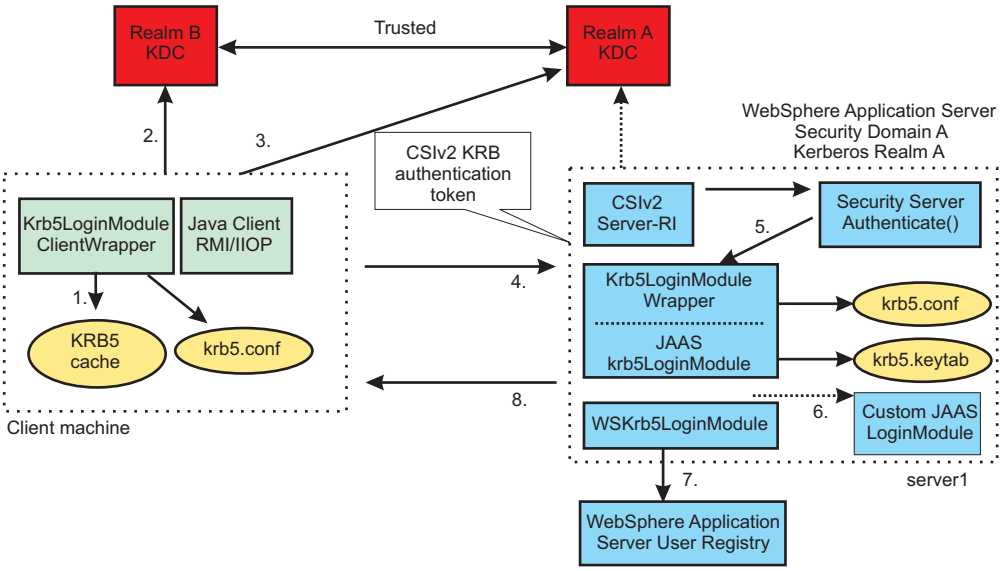


Figure 9. Using a Kerberos credential cache to authenticate to WebSphere Application Server with a Kerberos token in a trusted Kerberos realm

In the previous figure, the following events occur:

1. The client uses the Kerberos credential cache if it exists.
2. The client requests a cross realm ticket (TGS_REQ) for **`Realm A`** from the **`Realm B`** KDC using the Kerberos credential cache.
3. The client uses a cross realm ticket to request Kerberos service ticket for **`server1`** (TGS_REQ) from the **`Realm A`** KDC.
4. The Kerberos token returned from the KDC (TGS_REP) is added to the CSIv2 message authentication token and sent to **`server1`** for authentication.
5. The server calls Krb5LoginModuleWrapper to establish security context with the client using the server Kerberos Service Principal Name (SPN) and keys from the `krb5.keytab` file. If the server successfully establishes a security context with the client, it always extracts the client GSS delegation credential and tickets and places them in the client subject.
6. Optionally, a custom JAAS Login Module might be needed if the KDC and WebSphere Application Server do not use the same user registry.
7. The user is validated with the user registry for WebSphere Application Server.
8. The results (success or failure) are returned to the client.

The following figure shows a Java and administrative client that uses a Kerberos principal name and password to authenticate to WebSphere Application Server with a Kerberos token:

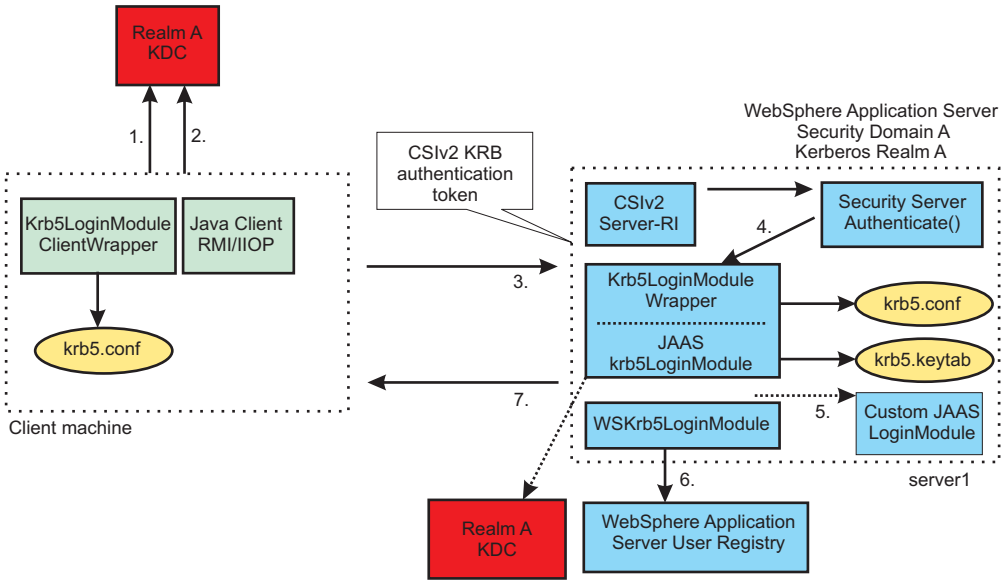


Figure 10. Using a Kerberos principal name and password to authenticate to WebSphere Application Server with a Kerberos token

In the previous figure, the following events occur:

1. The client obtains the Kerberos granting ticket (TGT) from the KDC.
2. The client obtains a Kerberos service ticket for **server1** (TGS_REQ) using the TGT.
3. The Kerberos token returned from the KDC (TGS_REP) is added to the CSIv2 message authentication token and sent to **server1** for authentication.
4. The server calls Krb5LoginModuleWrapper to establish security context with the client using the server Kerberos Service Principal Name (SPN) and keys from the `krb5.keytab` file. If the server successfully establishes a security context with the client, it always extracts the client GSS delegation credential and tickets and places them in the client subject.
5. Optionally, a custom JAAS Login Module might be needed if the KDC and WebSphere Application Server do not use the same user registry.
6. The user is validated with the user registry for WebSphere Application Server.
7. The results are returned to the client.

The following figure shows server-to-server communications:

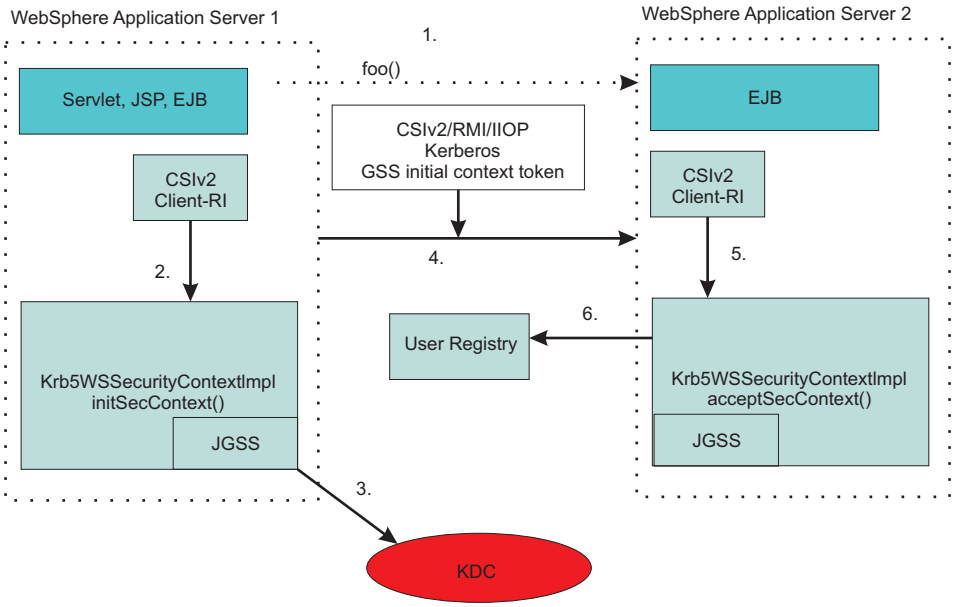


Figure 11. Server to server communications

When a WebSphere Application Server starts up, it uses the server ID and password to login to the KDC and then obtains the TGT. It then uses the TGT to request a service ticket to communicate with another server. If a WebSphere Application Server uses the internal server ID instead of the server ID and password, server-to-server communication is done using an LTPA token. In the previous figure, the following events occur:

1. WebSphere Application Server 1 invokes a method, foo(), on an Enterprise JavaBeans (EJB) running in WebSphere Application Server 2.
2. **Server1** obtains a Kerberos service ticket for **Server2** (TGS_REQ) using the **Server1** TGT.
3. Same as step 2.
4. The Kerberos token returned from a KDC (TGS_REP) is added to the CSIv2 message authentication token and sent to **Server2** for authentication.
5. **Server2** calls the acceptSecContext() method to establish security context with **server1** using the **server2** Kerberos Service Principal Name (SPN) and keys from the `krb5.keytab` file. If **server2** successfully establishes a security context with **server1**, it always extracts the **server1** GSS delegation credential and tickets and places them in the subject.
6. The server id is validated with the WebSphere user registry.

Note: If a Java client application and the application server exist on the same machine and they use different Kerberos realm names, the run time uses the default realm name from the Kerberos configuration file. Alternatively, you can specify the realm name during the login process.

Things to consider before setting up Kerberos as the authentication mechanism for WebSphere Application Server

WebSphere Application Server now supports SPNEGO tokens in the HTTP header, Kerberos tokens, LTPA tokens and BasicAuth (GSSUP) for authentication.

To provide end-to-end Kerberos and end-to-end SPNEGO to Kerberos solutions, be aware of the following:

- The `Enabled delegation of Kerberos` credentials option must be selected. Read about Configuring Kerberos as the authentication mechanism using the administrative console for more information about this option.

- A client must obtain a ticket-granting ticket (TGT) with forwardable, address-less and renewable flags so that a target server can extract a client delegation Kerberos credential and use it for going to the downstream server.
- A client TGT that has an address can not be used for a downstream server, Data replication service (DRS) cache and cluster environments.
- See your Kerberos KDC platforms to make sure that it allows for client delegation Kerberos.
- For a long running application, a client should request a TGT with a renewable flag so that a target server can renew the delegation Kerberos.
- For a long-running application, ensure that the Kerberos ticket is valid for a period of time that is at least as long as the application runs. For example, if the application processes a transaction that takes 5 minutes, the Kerberos ticket must be valid for at least 5 minutes.
- Kerberos authentication and SPNEGO web authentication are both supported for Active Directory cross domain trusts within the same forest.
- In order for an administrative agent to use the Kerberos authentication mechanism, it must exchange an LTPA key with an administrative subsystem profile.
- If you plan to use the client delegation Kerberos credential for downstream authentication, make sure the client can request a service ticket that is greater than 10 minutes. If the client delegation Kerberos credential lifetime is less than 10 minutes, then the server attempts to renew it.

Note: The client, WebSphere Application Server and KDC machines must keep the clock synchronized. The best practice is to use a time server to keep all of the systems synchronized.

For this release of WebSphere Application Server, be aware of the following:

- Complete end-to-end Kerberos support with Tivoli Access Manager is available using the following KDCs:
  - z/OS
  - Microsoft (single or multi-realm)
  - AIX
  - Linux
- You can now configure and enable Kerberos cross realms for WebSphere Application Server and the thin client.
- WebSphere Application Server administrative function with Kerberos is limited by the following:
  - The preferred authentication mechanism for flexible management activities is the Rivest Shamir Adleman (RSA) authentication mechanism (by default).
  - Job Manager configured with Kerberos as the administrative authentication does not support Cross-Kerberos realms. They must be in the same Kerberos realm as registered nodes, or have the administrative authentication set to RSA
  - While Kerberos authentication is supported for administrative clients (wsadmin or Java clients) you should use the same KDC realm as the WebSphere Application Server it administers. Otherwise, a user id and password are recommended.
  - Mixed cell Kerberos and LTPA configuration is not supported when some of the nodes are WebSphere Application Server Release 6.x nodes or earlier.

Support information for Kerberos authentication

The following scenarios are supported:

- External domain trusts that are not on the same forests
- Domain trust within the same forest
- Kerberos realm trust

The following scenarios are not supported:

- Cross-forest trust
- Forest external trusts

Setting up Kerberos as the authentication mechanism for WebSphere Application Server

You must perform the steps in order as listed in “Setting up Kerberos as the authentication mechanism for WebSphere Application Server” to set up Kerberos as the authentication mechanism for WebSphere Application Server.

Note: Kerberos authentication mechanism on the server side must be done by the system administrator and on the Java client side by end users. The Kerberos keytab file must to be protected.

Setting up Kerberos as the authentication mechanism for the pure Java client

End users can optionally set up Kerberos authentication mechanism for the pure Java client. Read about Configuring a Java client for Kerberos authentication for more information.

Setting up Kerberos as the authentication mechanism for WebSphere Application Server

You must perform steps in this article in order to set up Kerberos as the authentication mechanism for WebSphere Application Server.

About this task

Note: Kerberos authentication mechanism on the server side must be done by the system administrator and on the Java client side by end users. The Kerberos keytab file must to be protected.

You must first ensure that the KDC is configured. See your Kerberos Administrator and User's guide for more information.

gotcha: When configuring the envar file for a z/OS KDC, order the encryption types from most secure to least secure for the SKDC_TKT_ENCTYPES environment variable. The z/OS KDC prefers to use the encryption types that are first in the list, from left to right.

You must perform the following steps in order to set up Kerberos as the authentication mechanism for WebSphere Application Server.

Procedure

1. Create a Kerberos service principal name and keytab file You can create a Kerberos service principal name and keytab file using Microsoft Windows, iSeries, Linux, Solaris, Massachusetts Institute of Technology (MIT) and z/OS operating systems key distribution centers (KDCs).

   Kerberos prefers servers and services to have a host-based service ID. The format of this ID is `<service name>/<fully qualified hostname>`. The default service name is `WAS`. For Kerberos authentication, the service name can be any strings that are allowed by the KDC. However, for SPNEGO web authentication, the service name must be HTTP. An example of a WebSphere Application Sever server ID is `WAS/myhost.austin.ibm.com`.

   Each host must have a server ID unique to the hostname. All processes on the same node share the same host-based service ID.

   A Kerberos administrator creates a Kerberos service principal name (SPN) for each node in the WebSphere cell. For example, for a cell with 3 nodes (such as `server1.austin.ibm.com`, `server2.austin.ibm.com` and `server3.austin.ibm.com`), the Kerberos administrator must create the following Kerberos service principals: `WAS/server1.austin.ibm.com`, `WAS/server2.austin.ibm.com` and `WAS/server3.austin.ibm.com`.

The Kerberos keytab file (**krb5.keytab**) contains all of the SPNs for the node and must be protected. This file can be placed in the config/cells/<cell_name> directory

Read the Creating a Kerberos principal and keytab article for more information.

2. Create a Kerberos configuration file The IBM implementation of the Java Generic Security Service (JGSS) and KRB5 require a Kerberos configuration file (**krb5.conf** or **krb5.ini**) on each node or Java virtual machine (JVM). In this release of WebSphere Application Server, this configuration file should be placed in the config/cells/<cell_name> directory so that all application servers can access this file. If you do not have a Kerberos configuration file, use a **wsadmin** command to create one.

   Read the Creating a Kerberos configuration article for more information.

3. Configure Kerberos as the authentication mechanism for WebSphere Application Sever using the administrative console Use the administrative console to configure Kerberos as the authentication mechanism for the application server. When you have entered and applied the required information to the configuration, the Kerberos service principal name is formed as *<service name>/<fully qualified hostname>*@KerberosRealm, and is used to verify incoming Kerberos token requests.

   Read the Configuring Kerberos as the authentication mechanism using the administrative console article for more information.

4. Map a client Kerberos principal name to the WebSphere user registry ID You can map the Kerberos client principal name to the WebSphere user registry ID for both Simple and Protected GSS-API Negotiation (SPNEGO) web authentication and Kerberos authentication.

   Read the Mapping of a client Kerberos principal name to the WebSphere user registry ID article for more information.

5. Set up Kerberos as the authentication mechanism for the pure Java client (optional) A Java client can authenticate with WebSphere Application server with a Kerberos principal name and password or with the Kerberos credential cache (krb5Ccache).

   Read the Configuring a Java client for Kerberos authentication article for more information.

RSA token authentication mechanism

The Rivest Shamir Adleman (RSA) Authentication Mechanism is used to simplify the security environment for the Flexible Management Topology. It supports the ability to securely and easily register new servers to the Flexible Management topology. With the Flexible Management topology, you can submit and manage administrative jobs, locally or remotely, by using a job manager that manages applications, performs product maintenance, modifies configurations, and controls the application server runtime. The RSA authentication mechanism is only used for server-to-server administrative authentication, such as admin connector and file transfer requests. The RSA authentication mechanism does not replace LTPA or Kerboros for use by applications.

Note: The RSA token authentication mechanism aids the flexible management objective to preserve the base profiles configurations and isolate them from a security perspective. This mechanism permits the base profiles managed by an administrative agent to have different Lightweight Third-Party Authentication (LTPA) keys, different user registries, and different administrative users.

Important: The RSA token is not related to the RSA SecureId token. Please note that the application server does not provide support for SecureId.

Authentication is the process of establishing whether a client is who or what it claims to be in a particular context. A client can be either an end user, a machine, or an application. An authentication mechanism in WebSphere Application Server typically collaborates closely with a *user registry*. The user registry is the user and groups account repository that the authentication mechanism consults with when performing authentication. The authentication mechanism is responsible for creating a *credential*, which is an internal product representation of a successfully authenticated client user. Not all credentials are created equally. The abilities of the credential are determined by the configured authentication mechanism.

Authentication process

The RSA token authentication mechanism ensures that after the RSA root signer certificate (15 year lifetime) is exchanged between two administrative processes, there is no need to synchronize security information among disparate profiles for administrative requests. The RSA personal certificate (1 year lifetime) is used to perform the cryptographic operations on the RSA tokens and can be verified by the long-lived RSA root. RSA token authentication is different from LTPA where keys are shared and if one side changes, all sides need to change. Since RSA token authentication is based on a PKI infrastructure, it benefits from the scalability and manageability of this technology in a large topology.

An RSA token has more advanced security features than LTPA; this includes a nonce value that makes it a one-time use token, a short expiration period (since it's a one-time use token), and trust, which is established based on certificates in the target RSA trust store.

RSA token authentication does not use the same certificates as used by Secure Sockets Layer (SSL). This is the reason RSA has it's own keystores. In order to isolate the trust established for RSA, the trust store, keystore, and root keystore, need to be different from the SSL configuration.

Note: SSL personal certificates given to pure clients are often signed by the same SSL root certificate used by servers, and this allows a pure client to send an RSA token to a server and act as an administrator. This should be avoided for the RSA token authentication mechanism. The RSA token authentication mechanism has its own root certificate which signs personal certificates that are used to encrypt and sign parts of the token.

The data stored in an RSA token is based on the identity of the client subject. The client subject can be based on LTPA or Kerberos, but the RSA token does not use this protection for administrative requests. The RSA token is easier to use while still maintaining a secure transportation of the identity. The data in an RSA token includes:

- Version
- Nonce
- Expiration
- Realm
- Principal
- Access ID
- Roles (not currently used)
- Groups
- Custom data

Custom data can be added to the WSCredential on the sending side (prior to going outbound) by creating a properties object, adding custom attributes, and adding this to the WSCredential in the following way.

```
import com.ibm.websphere.security.cred.WSCredential;

java.util.Properties props = new java.util.Properties();
props.setProperty("myAttribute", "myValue");
WSCredential.put ("customRSAProperties", props);
```

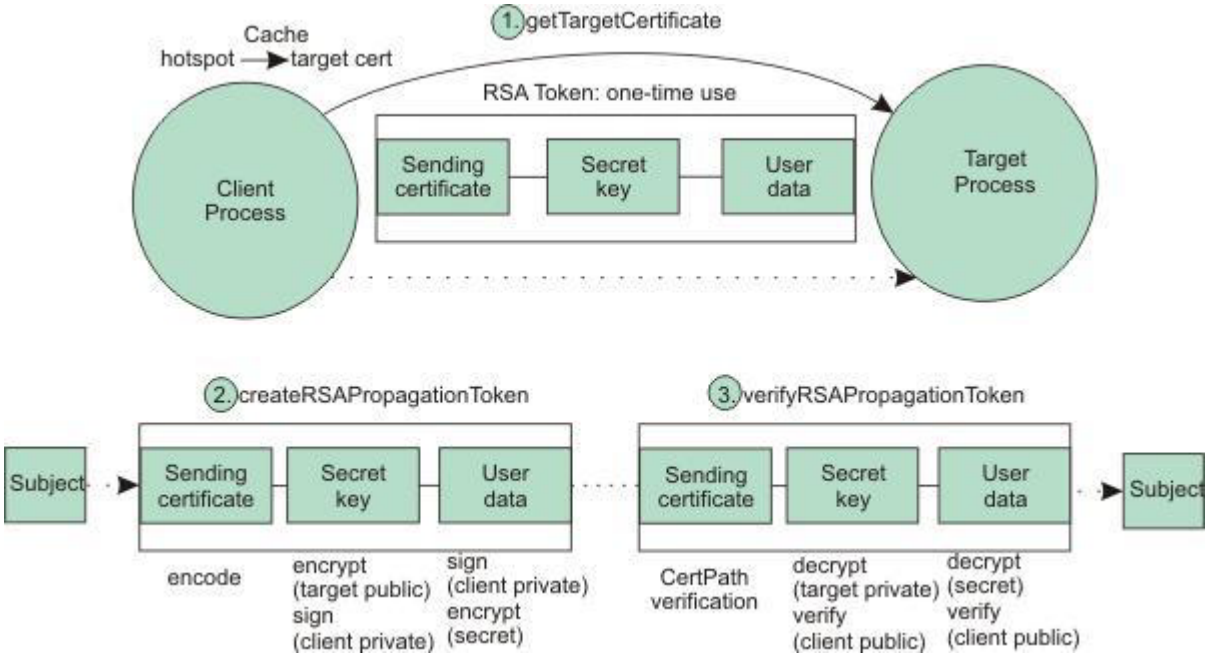
Once the Subject is created at the target process, you can get access to these attributes in the following way.

```
java.util.Properties props = (java.util.Properties) WSCredential.get("customRSAProperties");
```

This data is placed into a hash table at the target side and the hash table is used in a Java™ Authentication and Authorization Service (JAAS) login to obtain a subject at the target that contains the same attributes from the RSA token. With the target containing the same attributes from the RSA token,

you can have a subject at the target side that is not from the same realm used by the target. For this authorization to succeed, a cross-realm mapping is required within the administrative authorization table unless the identity is a trusted server ID.

The figure later in this section is an overview of the RSA token authentication mechanism and describes the process that takes place when a request is sent from a server-as-client to a target server. The server-as-client has an administrative subject on the thread that is used as input to create the RSA token. The other information needed is RSA public certificate of the target server. This certificate must be retrieved by making a “bootstrap” MBean request to the target process prior to sending any real requests. The target bootstrap request retrieves the public certificate from the target process. When creating an RSA token, the primary purpose of obtaining the target's public certificate is to encrypt the secret key. Only the target can decrypt the secret key, which is used to encrypt the user data.



The client's private key is used to sign both the secret key and the user data. The client's public key is embedded in the RSA token and validated at the target. If the client's public key is not trusted when calling the CertPath APIs at the target, the RSA token validation cannot continue. If the client's public key is trusted, it can be used to verify the secret key and user data signatures.

The basic goal is to convert the client subject into a subject at the target by securely propagating the required information. After the subject is generated at the target, the RSA authentication mechanism process is complete.

Configuring the RSA token authentication mechanism

You use the WebSphere Application Server administrative console to configure the Rivest Shamir Adleman (RSA) token authentication mechanism. The RSA token authentication mechanism can only be used for administrative requests. As such, the authentication mechanism choices for administrative authentication are part of the Global Security panel of the administrative console.

Before you begin

RSA token authentication mechanism is the default selection for the application server, administrative agent, and job manager profiles. LTPA is still the default for the deployment manager profile to preserve the same behavior for the existing topology.

About this task

You configure Lightweight Third-Party Authentication (LTPA) and Kerberos on the main authentication mechanism panels of the administrative console as well as configure RSA token authentication. During registration of a base profile with the administrative agent, the trusted certificates on both sides are updated with the root signer for the other. The same process occurs during registration of an administrative agent or deployment manager with a job manager. When removing the registration, the trusted signers are removed from both sides so that trust is no longer established.

By default, the RSA mechanism is set up correctly during the registration tasks, such as **registerNode** or **registerWithJobManager**. No further actions are necessary to establish trust within these environments. However, if you need to establish trust between two base servers or between two admin agents, for example, you can use the following steps to further configure the RSA token authentication mechanism:

Procedure

1. Click **Security > Global security** . Under Administrative security click the link to **Administrative authentication**.
2. Select the RSA token radio button. Select a data encryption keystore from the drop-down list. The option is recommend for flexible systems administration.
3. Optional: To exchange the root signers between two base servers:
   a. Select the root keystore from the Data encryption keystore drop-down list (such as NodeRSATokenRootStore).
   b. Click **Extract Signer**.
   c. Enter a fully-qualified name in the Certificate file name field.
   d. Click **OK**.
4. Optional: Transfer the extracted root signer to the other server, and add it to that server's trusted signers keystore:
   a. Select the trusted keystore from the drop-down list (such as NodeRSATokenTrustedStore).
   b. Click **Add Signer**.
   c. Enter a unique name for the Alias.
   d. Enter a fully-qualified name for the signer key file.
   e. Click **OK**.
5. Enter the nonce cache timeout value.
6. Enter token timeout value.
7. Click **Apply** and **Save**.

Results

You have configured the RSA token authentication mechanism.

RSA token authentication settings

Use this panel to configure RSA token authentication.

To view this administrative console page, click **Security > Global security**. Under Administrative security click **Administrative authentication**.

The administrative authentication method is used when an administrative process on this profile connects to another profile. If the primary authentication method is set to RSA token and that primary method fails, the system attempts to use the current application authentication method (which could be SWAM, Kerberos, or LTPA for example).

Note: SWAM is deprecated and will be removed in a future release.

RSA token (recommended for flexible systems administration):

RSA token is an authentication mechanism using certificates for signing and encryption portions of the security information being propagated.

Information	Value
Default:	Enabled

Data encryption keystore:

This is the keystore that contains the personal certificate used to encrypt and sign RSA tokens.

Information	Value
Data type:	text

Personal certificate for encryption:

This is the alias found in the Data encryption keystore that is used to encrypt and sign RSA tokens.

Information	Value
Data type:	text

Trusted signers keystore:

This is the keystore used to contain signer certificates that can validate RSA tokens sent by other servers. The RSA token contains a sending certificate that needs to be validated by this trust store using a CertPath validation.

Information	Value
Data type:	text

Nonce cache timeout:

Specifies the amount of time, in minutes, that the issued token is valid.

This field displays the maximum timeout, in minutes, for a token to be considered valid.

Information	Value
Data type:	Integer
Default:	20
Minimum:	10
Maximum:	Integer.MAX_VALUE

Token timeout:

Specifies the amount of time, in minutes, that the issued token is valid.

This field displays the maximum timeout, in minutes, for a token to be considered valid.

Information	Value
Data type:	Integer
Default:	10
Minimum:	10
Maximum:	Integer.MAX_VALUE

Only use the active application authentication mechanism (currently LTPA):

Select to encrypt authentication information so that the application server can send the data from one server to another in a secure manner.

The encryption of authentication information that is exchanged between servers involves the Lightweight Third-Party Authentication (LTPA) mechanism.

Kerberos:

Select to encrypt authentication information so that the application server can send the data from one server to another in a secure manner.

The encryption of authentication information that is exchanged between servers involves the Kerberos mechanism.

Note: Kerberos must be configured before this option can be selected.

RSA token certificate use

The Rivest Shamir Adleman (RSA) token uses certificates in a similar way that Secure Sockets Layer (SSL) uses them. However, the trust established for SSL and RSA are different, and RSA certificates should not use SSL certificates and vice versa. The SSL certificates can be used by pure clients, and when used for the RSA mechanism would allow the client to send an RSA token to the server. The RSA token authentication mechanism is purely for server-to-server requests and should not be used by pure clients. The way to prevent this is to control the certificates used by RSA in such as a way so they are never distributed to any clients. There is a different root certificate for RSA that prevents trust being established with clients who only need SSL certificates.

RSA root certificate

For each profile there is a root certificate stored in the `rsatoken-root-key.p12` keystore. The sole purpose of this RSA root certificate is to sign the RSA personal certificate which is stored in the `rsatoken-key.p12` keystore. The RSA root certificate has a default lifetime of 15 years. The signer from the RSA root certificate is shared with other processes to establish trust.

The keytool utility is available using the QShell Interpreter. Using the keytool utility, you can list the contents of these keystores and display the keyEntry (personal certificate). The following example illustrates how this is accomplished for the `rsatoken-root-key.p12` (RSA root certificate) and `rsatoken-key.p12` (RSA personal certificate).

```
${profile_root}\config\cells\${cellname}\nodes\${nodename}> keytool -list -v -keystore rsatoken-root-key.p12
–storepass WebAS -storetype PKCS12

Alias name: root
Entry type: keyEntry
Certificate[1]:
Owner: CN=9.41.62.64, OU=Root Certificate, OU=BIRKT60AACell04, OU=BIRKT60AANode04, O=IBM, C=US
Issuer: CN=9.41.62.64, OU=Root Certificate, OU=BIRKT60AACell04, OU=BIRKT60AANode04, O=IBM, C=US
Serial number: 3474fccaf789d
Valid from: 11/12/07 2:50 PM until: 11/7/27 2:50 PM
```

```
Certificate fingerprints:
         MD5:  7E:E6:C7:E8:40:4E:9B:96:5A:66:E5:0B:37:0B:08:FD
         SHA1: 36:94:81:55:C4:48:83:27:89:C7:16:D2:AD:3D:3E:67:DF:1D:6E:87

${profile_root}\config\cells\${cellname}\nodes\${nodename}> keytool -list -v -keystore rsatoken-key.p12
–storepass WebAS -storetype PKCS12

Alias name: default
Entry type: keyEntry
Certificate[1]:
Owner: CN=9.41.62.64, OU=BIRKT60AACell04, OU=BIRKT60AANode04, O=IBM, C=US
Issuer: CN=9.41.62.64, OU=Root Certificate, OU=BIRKT60AACell04, OU=BIRKT60AANode04, O=IBM, C=US
Serial number: 3475073488921
Valid from: 11/12/07 2:50 PM until: 11/11/08 2:50 PM
Certificate fingerprints:
         MD5:  FF:1C:42:E3:DA:FF:DC:A4:35:B2:33:30:D1:6E:E0:19
         SHA1: A4:FB:9D:7B:A1:5B:6A:37:9F:20:BD:B2:BD:98:FA:68:71:57:28:62
Certificate[2]:
Owner: CN=9.41.62.64, OU=Root Certificate, OU=BIRKT60AACell04, OU=BIRKT60AANode04, O=IBM, C=US
Issuer: CN=9.41.62.64, OU=Root Certificate, OU=BIRKT60AACell04, OU=BIRKT60AANode
04, O=IBM, C=US
Serial number: 3474fccaf789d
Valid from: 11/12/07 2:50 PM until: 11/7/27 2:50 PM
Certificate fingerprints:
         MD5:  7E:E6:C7:E8:40:4E:9B:96:5A:66:E5:0B:37:0B:08:FD
         SHA1: 36:94:81:55:C4:48:83:27:89:C7:16:D2:AD:3D:3E:67:DF:1D:6E:87
```

The purpose of the RSA personal certificate is to sign and encrypt information in the RSA token. The RSA personal certificate has a default lifetime of one year because it is used to sign and encrypt data that is transmitted over the wire. Refreshing the certificate is performed by the certificate expiration monitor, which is used for any other certificate in the system including SSL certificates.

RSA token trust is established when the `rsatoken-trust.p12` of the target process contains the signer of the root certificate of the client process that sends a token. Inside the RSA token is the public certificate of the client, which must be validated at the target before being used to decrypt data. The validation of the client's public certificate is performed using the CertPath APIs, which use the **rsatoken-trust.p12** as the source of certificates used during the validation.

The following example shows the use of the keytool utility to list the `rsatoken-trust.p12` keystore.

Note: This trust store contains three trustedCertEntry (public certificate) entries. The root public certificate from the administrative agent, a root public certificate from a job manager to which it is registered, and a root public certificate from a base profile to which it is registered.

```
${profile_root}\config\cells\${cellname}\nodes\${nodename}> keytool -list -v -keystore rsatoken-trust.p12
–storepass WebAS -storetype PKCS12

Alias name: root
Entry type: trustedCertEntry

Owner: CN=9.41.62.64, OU=Root Certificate, OU=BIRKT60AACell04, OU=BIRKT60AANode04, O=IBM, C=US
Issuer: CN=9.41.62.64, OU=Root Certificate, OU=BIRKT60AACell04, OU=BIRKT60AANode04, O=IBM, C=US
Serial number: 3474fccaf789d
Valid from: 11/12/07 2:50 PM until: 11/7/27 2:50 PM
Certificate fingerprints:
         MD5:  7E:E6:C7:E8:40:4E:9B:96:5A:66:E5:0B:37:0B:08:FD
         SHA1: 36:94:81:55:C4:48:83:27:89:C7:16:D2:AD:3D:3E:67:DF:1D:6E:87
*******************************************
Alias name: cn=9.41.62.64, ou=root certificate, ou=birkt60jobmgrcell02, ou=birkt
60jobmgr02, o=ibm, c=us
Entry type: trustedCertEntry

Owner: CN=9.41.62.64, OU=Root Certificate, OU=BIRKT60JobMgrCell02, OU=BIRKT60JobMgr02, O=IBM, C=US
Issuer: CN=9.41.62.64, OU=Root Certificate, OU=BIRKT60JobMgrCell02, OU=BIRKT60JobMgr02, O=IBM, C=US
```

```
Serial number: 34cc4c5d71740
Valid from: 11/12/07 4:30 PM until: 11/7/27 4:30 PM
Certificate fingerprints:
         MD5:  AB:65:3A:04:5B:C7:6D:A8:B1:98:B9:7B:65:A8:FA:F8
         SHA1: C0:83:FE:D0:B6:30:FB:A1:10:41:4B:8E:50:4B:78:40:0F:E5:E3:35
*******************************************
Alias name: birkt60node19_signer
Entry type: trustedCertEntry

Owner: CN=9.41.62.64, OU=Root Certificate, OU=BIRKT60Node15Cell, OU=BIRKT60Node19, O=IBM, C=US
Issuer: CN=9.41.62.64, OU=Root Certificate, OU=BIRKT60Node15Cell, OU=BIRKT60Node19, O=IBM, C=US
Serial number: 34825d997fda3
Valid from: 11/12/07 3:06 PM until: 11/7/27 3:06 PM
Certificate fingerprints:
         MD5:  66:61:CE:7C:C7:44:8B:A7:23:FF:1B:68:E4:AC:24:55
         SHA1: 25:E0:6B:D9:60:BB:67:5B:C6:67:BD:02:2C:54:E3:DA:24:E5:31:A3
*******************************************
```

You can use the WebSphere Application Server certificate management tools to create a new personal certificate, and then replace the RSA personal certificate in the rsa-key.p12 and the public key in the rsa-trust.p12 with this newly created personal certificate. If you replace the RSA personal certificate prior to federation to an administrative agent or job manager, the exchange of certificates is done for you. If you change the certificate after federation, you need to make sure the rsa-trust.p12 on the administrative agent or job manager is updated with the signer for your new certificate to establish trust.

Simple WebSphere authentication mechanism (deprecated)

The Simple WebSphere authentication mechanism (SWAM) defines rules about security information and the format of how security information is stored in both credentials and tokens. SWAM is intended for simple, non-distributed, single application server runtime environments.

Note: SWAM was deprecated in WebSphere Application Server Version 8.5 and will be removed in a future release.

The single application server restriction is due to the fact that SWAM does not support *forwardable* credentials. If a servlet or enterprise bean in application server process 1, invokes a remote method on an enterprise bean living in another application server process 2, the identity of the caller identity in process 1 is not transmitted to server process 2. What is transmitted is an unauthenticated credential, which, depending on the security permissions configured on the EJB methods, can cause authorization failures.

Because SWAM is intended for a single application server process, single sign-on (SSO) is not supported.

The SWAM authentication mechanism is suitable for simple environments, software development environments, or other environments that do not require a distributed security solution.

Message layer authentication

Defines the credential information and sends that information across the network so that a receiving server can interpret it.

When you send authentication information across the network using a token the transmission is considered message layer authentication because the data is sent with the message inside a service context.

A pure Java client uses Kerberos (KRB5) or basic authentication, or Generic Security Services Username Password (GSSUP), as the authentication mechanism to establish client identity.

However, a servlet can use either basic authentication (GSSUP) or the authentication mechanism of the server, Kerberos (KRB5) or Lightweight Third Party Authentication (LTPA), to send security information in the message layer. Use KRB5 or LTPA by authenticating or by mapping the basic authentication credentials to the security mechanism of the server.

The security token that is contained in a token-based credential is authentication mechanism-specific. The way that the token is interpreted is only known by the authentication mechanism. Therefore, each authentication mechanism has an object ID (OID) representing it. The OID and the client token are sent to the server, so that the server knows which mechanism to use when reading and validating the token. The following list contains the OIDs for each mechanism:

BasicAuth (GSSUP): oid:2.23.130.1.1.1
KRB5: OID: 1.2.840.113554.1.2.2
LTPA: oid:1.3.18.0.2.30.2
SWAM: No OID because it is not forwardable

Note: SWAM is deprecated in WebSphere Application Server Version 8.5 and will be removed in a future release.

On the server, the authentication mechanisms can interpret the token and create a credential, or they can authenticate basic authentication data from the client, and create a credential. Either way, the created credential is the `received` credential that the authorization check uses to determine if the user has access to invoke the method. You can specify the authentication mechanism by using the following property on the client side:

- com.ibm.CORBA.authenticationTarget

Basic authentication (BasicAuth) and KRB5 are currently the only valid values. You can configure the server through the administrative console.

Note: When perform basic authentication is enabled, if the client is not similarly configured (and does not pass a credential such as a user ID and password).

Configuring authentication retries

Situations occur where you want a prompt to display again if you entered your user ID and password incorrectly or you want a method to retry when a particular error occurs back at the client. If you can correct the error by information at the client side, the system automatically performs a retry without the client seeing the failure, if the system is configured appropriately.

Some of these errors include:
- Entering a user ID and password that are not valid
- Having an expired credential on the server
- Failing to find the stateful session on the server

By default, authentication retries are enabled and perform three retries before returning the error to the client. Use the com.ibm.CORBA.authenticationRetryEnabled property (`True` or `False`) to enable or disable authentication retries. Use the com.ibm.CORBA.authenticationRetryCount property to specify the number of retry attempts.

Immediate validating of a basic authentication login

In WebSphere Application Server Version 6.x, a behavior is defined during request_login for a BasicAuth login. In releases prior to Version 5, a BasicAuth login takes the user ID and password entered through the loginSource method and creates a BasicAuth credential. If either the user ID or the password is not valid, the client program does not find out until the first method request is attempted. When the user ID or password is specified during a prompt or programmatic login, the user ID and password are authenticated by default with the security server, with a `True` or `False` returned as the result. If `False`, an

`org.omg.SecurityLevel2.LoginFailed` exception is returned to the client indicating that the user ID and password are not valid. If `True`, then the BasicAuth credential is returned to the caller of the request_login. To disable this feature on the pure client, specify `com.ibm.CORBA.validateBasicAuth=false`. By default, this feature is set to `True`. On the server side, specify this property in the security dynamic properties.

Integrating third-party HTTP reverse proxy servers

These steps are required to use a trust association interceptor with a reverse proxy security server.

About this task

WebSphere Application Server enables you to use multiple trust association interceptors. The application server uses the first interceptor that can handle the request.

Procedure

1. Access the administrative console.

   Type `http://`*`fully_qualified_host_name`*`:`*`port_number`*`/ibm/console` in a web browser.

   Port 9060 is the default port number for accessing the administrative console. During installation, however, you might have specified a different port number. Use the appropriate port number.
2. Click **Security > Global security**.
3. Under Web and SIP security, click **Trust association**.
4. Select the **Enable trust association** option.
5. Under Additional properties, click **Interceptors**. The default value appears.
6. Verify that the appropriate trust association interceptors are listed.

Results

Trust association is enabled.

What to do next

1. If you are enabling security, make sure that you complete the remaining steps for enabling security.
2. Save, stop and restart all of the product servers (deployment managers, nodes and application servers) for the changes to take effect.

Trust associations

`Trust association` enables the integration of IBM WebSphere Application Server security and third-party security servers. More specifically, a reverse proxy server can act as a front-end authentication server while the product applies its own authorization policy onto the resulting credentials that are passed by the proxy server.

Demand for such an integrated configuration has become more compelling, especially when a single product cannot meet all of the customer needs or when migration is not a viable solution. This article provides a conceptual background behind the approach.

In this setup, WebSphere Application Server is used as a back-end server to further exploit its fine-grained access control. The reverse proxy server passes the HTTP request to WebSphere Application Server that includes the credentials of the authenticated user. WebSphere Application Server then uses these credentials to authorize the request.

Trust association model

The idea that WebSphere Application Server can support trust association implies that the product application security recognizes and processes HTTP requests that are received from a reverse proxy

server. WebSphere Application Server and the proxy server engage in a contract in which the product gives its full trust to the proxy server and the proxy server applies its authentication policies on every web request that is dispatched to WebSphere Application Server. This trust is validated by the interceptors that reside in the product environment for every request received. The method of validation is agreed upon by the proxy server and the interceptor.

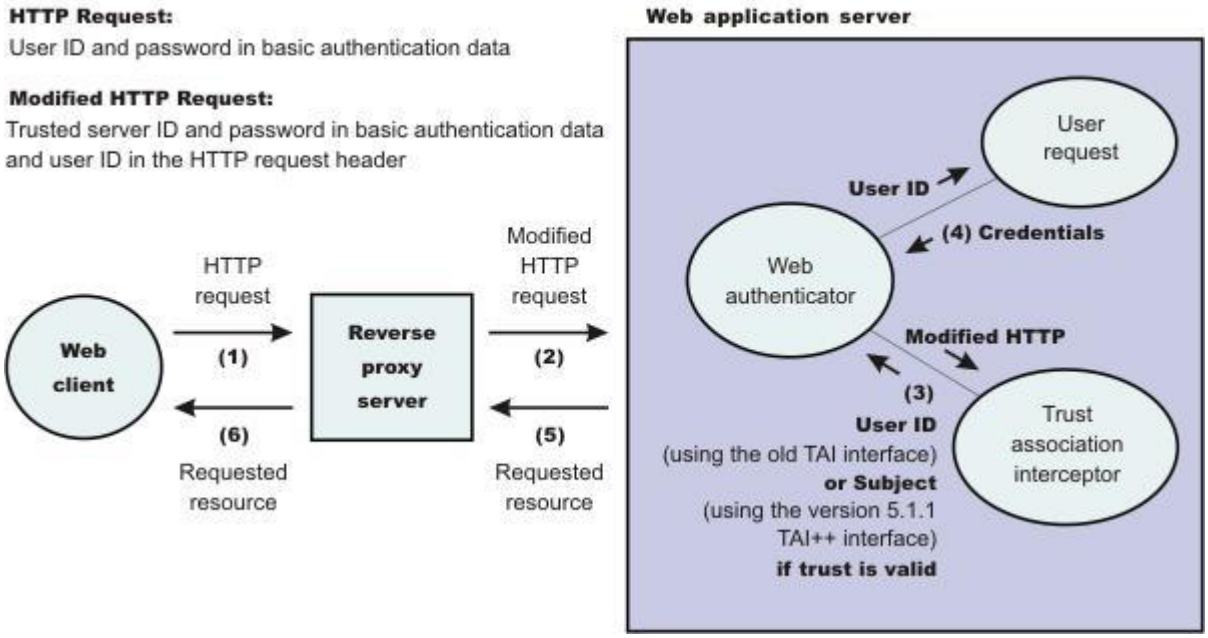
Running in trust association mode does not prohibit WebSphere Application Server from accepting requests that did not pass through the proxy server. In this case, no interceptor is needed for validating trust.

WebSphere Application Server supports the following trust association interceptor (TAI) interfaces:

com.ibm.ws.security.web.TAMTrustAssociationInterceptorPlus
This TAI interceptor implementation that implements the new WebSphere Application Server interface supports WebSphere Application Server Version 5.1.1 and later. The interface supports WebSEAL Version 5.1, but does not support WebSEAL Version 4.1. For an explanation of security attribute propagation, see “Security attribute propagation” on page 468.

com.ibm.ws.security.spnego.TrustAssociationInterceptorImpl
This interceptor is new to this release. SPNEGO has replaced SPNEGO TAI as the web authenticator for WebSphere Application Server.

Trust association model



IBM WebSphere Application Server: WebSEAL Integration

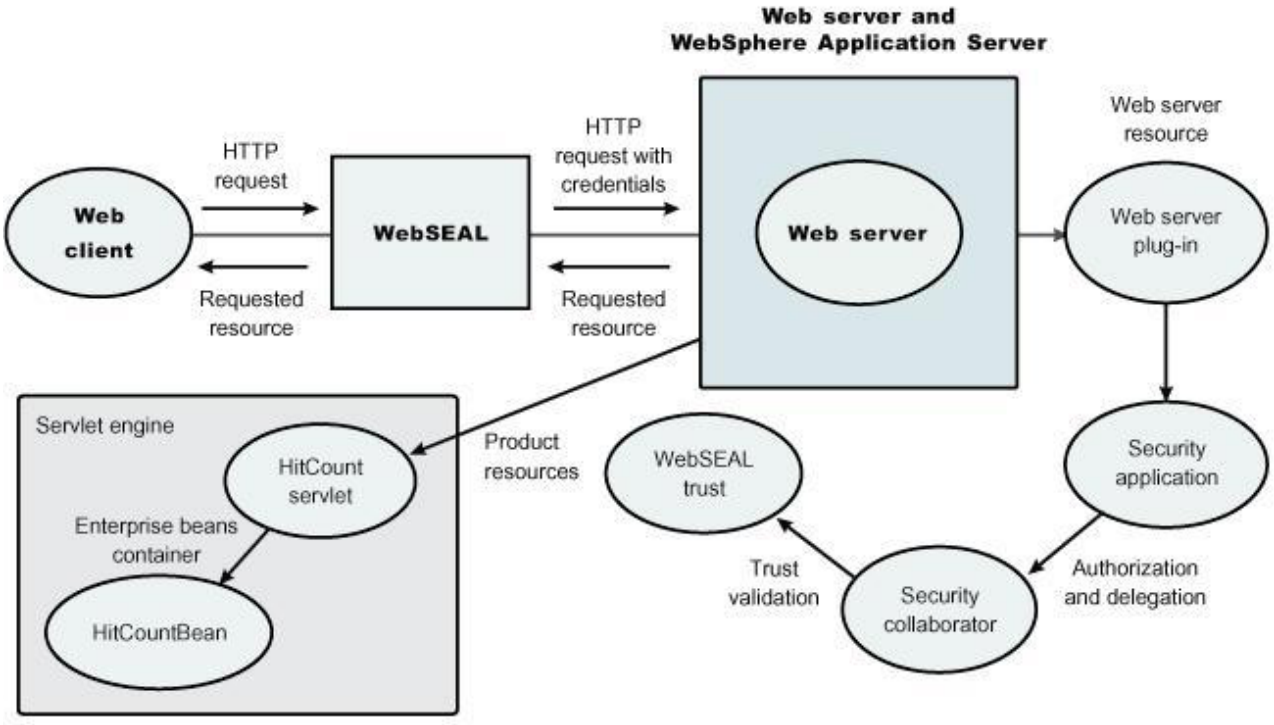
The integration of WebSEAL and WebSphere Application Server security is achieved by placing the WebSEAL server at the front-end as a reverse proxy server. From a WebSEAL management perspective, a junction is created with WebSEAL on one end, and the product web server on the other end. A junction is a logical connection that is created to establish a path from the WebSEAL server to another server.

In this setup, a request for web resources that are stored in a protected domain of the product is submitted to the WebSEAL server where it is authenticated against the WebSEAL security realm. If the requesting user has access to the junction, the request is transmitted to the WebSphere Application Server HTTP server through the junction, and then to the application server.

Meanwhile, WebSphere Application Server validates every request that comes through the junction to ensure that the source is a trusted party. This process is referenced as **validating the trust** and it is performed by a WebSEAL product-designated interceptor. If the validation is successful, WebSphere Application Server authorizes the request by checking whether the client user has the required permissions to access the web resource. If so, the web resource is delivered to the WebSEAL server through the web server, which then gives the resource to the client user.

WebSEAL server

The policy director delegates all of the web requests to its web component, the WebSEAL server. One of the major functions of the server is to perform authentication of the requesting user. The WebSEAL server consults a Lightweight Directory Access Protocol (LDAP) directory. It can also map the original user ID to another user ID, such as when global single sign-on (GSO) is used.

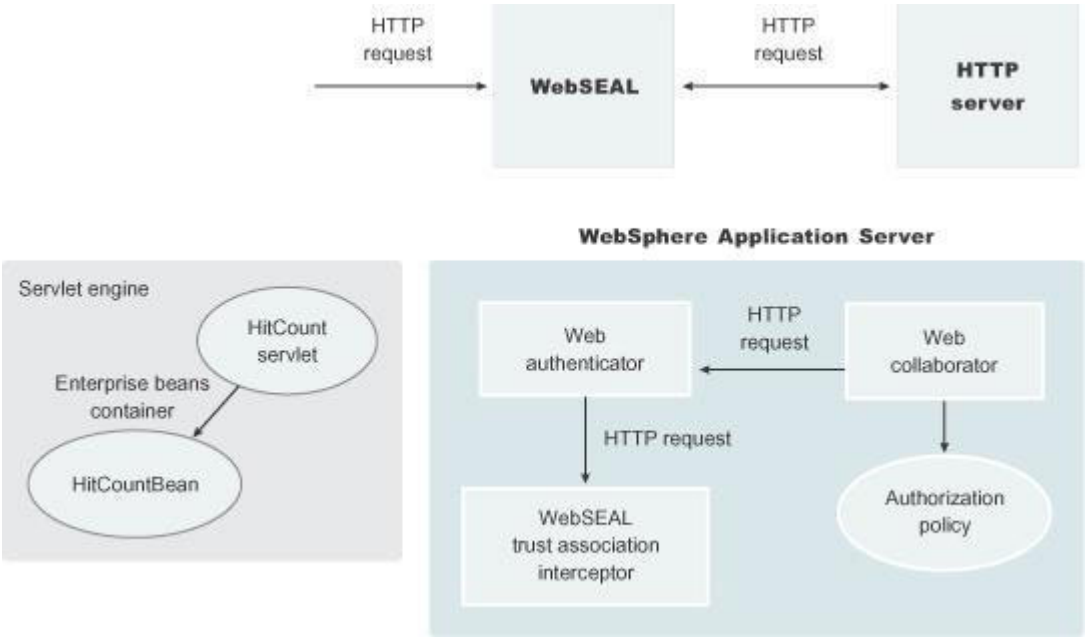


For successful authentication, the server plays the role of a client to WebSphere Application Server when channeling the request. The server needs its own user ID and password to identify itself to WebSphere Application Server. This identity must be valid in the security realm of WebSphere Application Server. The WebSEAL server replaces the basic authentication information in the HTTP request with its own user ID and password. In addition, WebSphere Application Server must determine the credentials of the requesting client so that the application server has an identity to use as a basis for its authorization decisions. This information is transmitted through the HTTP request by creating a header called **iv-creds**, with the Tivoli Access Manager user credentials as its value.

HTTP server

The junction that is created in the WebSEAL server must get to the HTTP server that serves as the product front end. However, the HTTP server is shielded from knowing that trust association is used. As far as it is concerned, the WebSEAL product is just another HTTP client, and as part of its normal routines, it sends the HTTP request to the product. The only requirement on the HTTP server is a Secure

Sockets Layer (SSL) configuration using server authentication only. This requirement protects the requests that flow within the junction.



Web collaborator

When trust association is enabled, the web collaborator manages the interceptors that are configured in the system. The web collaborator loads and initializes these interceptors when you restart your servers. When a request is passed to WebSphere Application Server by the Web server, the web collaborator eventually receives the request for a security check. Two actions must take place:

1. The request must be authenticated.
2. The request must be authorized.

The web authenticator is called to authenticate the request by passing the HTTP request. If successful, a good credential record is returned by the authenticator, which the web collaborator uses to base its authorization for the requested resource. If the authorization succeeds, the web collaborator indicates to WebSphere Application Server that the security check has succeeded and that the requested resource can be served.

Web authenticator

The web authenticator is asked by the web collaborator to authenticate a given HTTP request. Knowing that trust association is enabled, the task of the web authenticator is to find the appropriate trust association interceptor to direct the request for processing. The web authenticator queries every available interceptor. If no target interceptor is found, the web authenticator processes the request as though trust association is not enabled.

Note:

> WebSphere Application Server Version 4 through WebSphere Application Server Version 6.x support the com.ibm.websphere.security.TrustAssociationInterceptor.java interface. WebSphere Application Server Version 7.0.x and later supports the com.ibm.ws.security.spnego.TrustAssociationInterceptorImpl interface.

Trust association interceptor interface

The intent of the trust association interceptor interface is to have reverse proxy security servers (RPSS) exist as the exposed entry points to perform authentication and coarse-grained authorization, while WebSphere Application Server enforces further fine-grained access control. Trust associations improve security by reducing the scope and risk of exposure.

In a typical e-business infrastructure, the distributed environment of a company consists of web application servers, web servers, existing systems, and one or more RPSS, such as the Tivoli WebSEAL product. Such reverse proxy servers, front-end security servers, or security plug-ins registered within web servers, guard the HTTP access requests to the web servers and the web application servers. While protecting access to the Uniform Resource Identifiers (URIs), these RPSS perform authentication, coarse-grained authorization, and request routing to the target application server.

When a web server, such as an IBM HTTP Server, uses a TAI to communicate with WebSphere Application Server, sometimes it is essential for the TAI to know whether a request came through a web server or came directly to WebSphere Application Sever. Therefore the WebSphere Application Server Web container uses three HttpServletRequest attributes to provide the TAI with the certificate information for a request:

- The com.ibm.websphere.ssl.direct_connection_peer_certificates attribute contains a X509Certificate[] object of the certificate for a direct peer.
- The com.ibm.websphere.ssl.direct_connection_cipher_suite attribute contains a string object of a direct cipher suite.
- The com.ibm.websphere.webcontainer.is_direct_connection attribute contains a boolean object that indicates whether the connection was made through a web server, or was made directly to WebSphere Application Server.

See the topic *Web container request attributes* for more information about these attributes.

Trust association settings

Use this page to enable trust association, which integrates application server security and third-party security servers. More specifically, a reverse proxy server can act as a front-end authentication server while the product applies its own authorization policy onto the resulting credentials passed by the proxy server.

To view this administrative console page, complete the following steps:

1. Click **Security > Global security**.
2. Under Authentication, expand Web security and click **Trust association**.

When security is enabled and any of these properties change, go to the Global security panel and click **Apply** to validate the changes.

Enable trust association

Specifies whether trust association is enabled.

Information	Value
Data type:	Boolean
Default:	Disable
Range:	Enable or Disable

Trust association interceptor collection

Use this page to specify trust information for reverse security proxy servers.

To view this administrative console page, complete the following steps:

1. Click **Security > Global security**.
2. Under Authentication, expand Web and SIP security and click **Trust association**.
3. Under Additional Properties, click **Interceptors**.

When security is enabled and any of these properties are changed, go to the Global security panel and click **Apply** to validate the changes.

Interceptor class name

Specifies the trust association interceptor class name.

Data type

String

Trust association interceptor settings

Use this page to specify trust information for reverse security proxy servers.

To view this administrative console page, complete the following steps:

1. Click **Security > Global security**.
2. Under Authentication, click **Web and SIP security**.
3. Click **Trust association**.
4. Under Additional Properties, click **Interceptors > New**.

Interceptor class name

Specifies the trust association interceptor class name.

Data type

String

Single sign-on for authentication

With single sign-on (SSO) support, web users can authenticate once when accessing both WebSphere Application Server resources, such as HTML, JavaServer Pages (JSP) files, servlets, enterprise beans, and Lotus Domino resources, such as documents in a Domino database, or accessing resources in multiple WebSphere Application Server domains.

There are various ways to accomplish SSO, with the most common in WebSphere using LTPA cookies. LTPA cookies do not require any particular client and allow SSO across different cells provide the registry and LTPA keys are the same.

There are other flavors of SSO, including Simple and Protected GSS-API Negotiation (SPNEGO), which is a way to use the token from a Kerberos login (typically Windows) to authenticate to WebSphere Application Server. This prevents the user from having to type in their userid and passwords again.

Note: In WebSphere Application Server Version 6.1, a trust association interceptor (TAI) that uses the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) to securely negotiate and authenticate HTTP requests for secured resources was introduced. This function was deprecated In WebSphere Application Server 7.0. SPNEGO web authentication has taken its place to provide dynamic reload of the SPNEGO filters and to enable fallback to the application login method.

TAIs are also a form of single sign-on when used in combination with a Proxy server that does the front-end authentication. The TAI allows the credentials to flow to WebSphere from the Proxy server and to be used to login without the need to re-authenticate the user.

Single sign-on for authentication using LTPA cookies

With single sign-on (SSO) support, web users can authenticate once when accessing both WebSphere Application Server resources, such as HTML, JavaServer Pages (JSP) files, servlets, enterprise beans, and Lotus Domino resources, such as documents in a Domino database, or accessing resources in multiple WebSphere Application Server domains.

Application servers distributed in multiple nodes and cells can securely communicate using the Lightweight Third Party Authentication (LTPA) protocol. LTPA is intended for distributed, multiple application server and machine environments. LTPA can support security in a distributed environment through cryptography. This support permits LTPA to encrypt, digitally sign, and securely transmit authentication-related data, and later decrypt and verify the signature.

LTPA also provides the SSO feature wherein a user is required to authenticate only once in a domain name system (DNS) domain and can access resources in other WebSphere Application Server cells without getting prompted. Web users can authenticate once to a WebSphere Application Server or to a Domino server. This authentication is accomplished by configuring WebSphere Application Servers and the Domino servers to share authentication information.

Without logging in again, web users can access other WebSphere Application Servers or Domino servers in the same DNS domain that are enabled for SSO. You can enable SSO among WebSphere Application Servers by configuring SSO for WebSphere Application Server. To enable SSO between WebSphere Application Servers and Domino servers, you must configure SSO for both WebSphere Application Server and for Domino.

Prerequisites and conditions

To take advantage of support for SSO between WebSphere Application Servers or between WebSphere Application Server and a Domino server, applications must meet the following prerequisites and conditions:

- Verify that all servers are configured as part of the same DNS domain. The realm names on each system in the DNS domain are case sensitive and must match identically. For example, if the DNS domain is specified as `mycompany.com`, then SSO is effective with any Domino server or WebSphere Application Server on a host that is part of the `mycompany.com` domain, for example, `a.mycompany.com` and `b.mycompany.com`.
- Verify that all servers share the same registry.

  This registry can be either a supported Lightweight Directory Access Protocol (LDAP) directory server or, if SSO is configured between two WebSphere Application Servers, a stand-alone custom registry.Domino servers do not support stand-alone custom registries, but you can use a Domino-supported registry as a stand-alone custom registry within WebSphere Application Server.

  You can use a Domino directory that is configured for LDAP access or other LDAP directories for the registry. The LDAP directory product must have WebSphere Application Server support. Supported products include both Domino and LDAP servers, such as IBM Tivoli Directory Server. Regardless of the choice to use an LDAP or a stand-alone custom registry, the SSO configuration is the same. The difference is in the configuration of the registry.
- Define all users in a single LDAP directory. Using multiple Domino directory assistance documents to access multiple directories also is not supported.
- Enable HTTP cookies in browsers because the authentication information that is generated by the server is transported to the browser in a cookie. The cookie is used to propagate the authentication information for the user to other servers, exempting the user from entering the authentication information for every request to a different server.
- For a Domino server:
  - Domino Release 6.5.4 for iSeries and other platforms are supported.

  - A Lotus Notes® client Release 5.0.5 or later is required for configuring the Domino server for SSO.
  - You can share authentication information across multiple Domino domains.
- For WebSphere Application Server:
  - WebSphere Application Server Version 3.5 or later for all platforms are supported.
  - You can use any HTTP web server that is supported by WebSphere Application Server.
  - You can share authentication information across multiple product administrative domains.
  - Basic authentication (user ID and password) using the basic and form-login mechanisms is supported.

    Note: Form-login mechanisms for web applications require that SSO is enabled.
  - By default, WebSphere Application Server does a case-sensitive comparison for authorization. This comparison implies that a user who is authenticated by Domino matches the entry exactly (including the base distinguished name) in the WebSphere Application Server authorization table. If case sensitivity is not considered for the authorization, enable the **Ignore Case** property in the LDAP user registry settings.

Using a WebSphere Application Server API to achieve downstream web single sign-on with an LtpaToken2 cookie

You can programmatically perform downstream Single Sign On (SSO) web propagation of a Lightweight Third Party Authentication (LTPA) cookie without the need for an application to store and send user credentials.

WebSphere Application Server provides API support to propagate an LtpaToken2 cookie to downstream web single sign-on applications.

Web applications running in mid-tier WebSphere servers might need to propagate LtpaToken2 cookies on downstream web invocations. In this release of WebSphere Application Server, a new Application Programming Interface (API) is provided for application developers to programmatically perform downstream SSO without the need for an application to store and send user credentials.

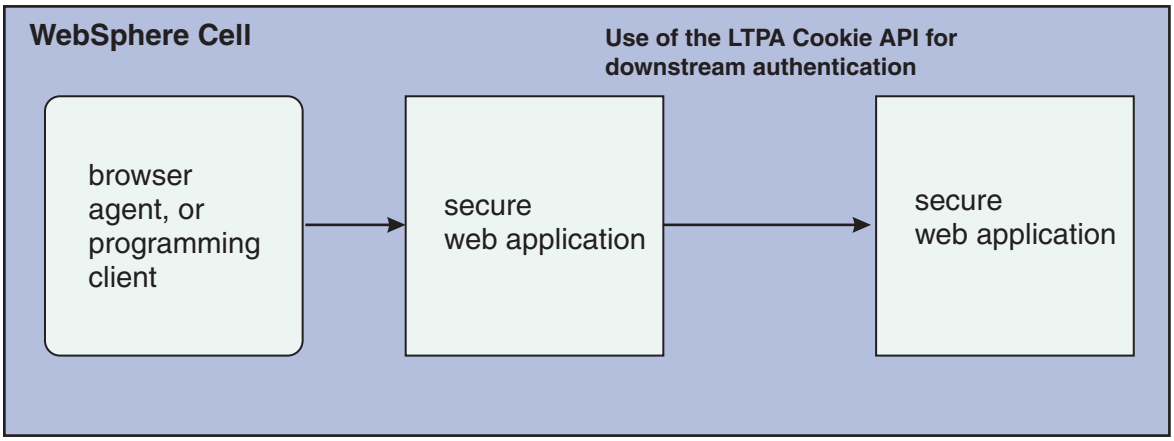


Figure 12. Use of the LTPA Cookie API for downstream authentication

This function is a public API in package com.ibm.websphere.security.WSSecurityHelper, and is defined as follows:

```
/**
   * Extracts an LTPA sso token from the subject of current
   * thread and builds a ltpa cookie out of it for use on
   * downstream web invocations.
   * When the returned value is not null use Cookie methods
   * getName() and getValue() to set the Cookie header
```

```
   * on an http request with header value of
   * Cookie.getName()=Cookie.getValue()
   *
   * @return an object of type javax.servlet.http.Cookie.
   *
   */
```

The following is an example of how you can use the new WSSecurityHelper API:

```
import javax.servlet.http.Cookie;
import com.ibm.websphere.security.WSSecurityHelper;

Cookie ltpaCookie = WSSecurityHelper.getLTPACookieFromSSOToken()
```

Note: The getLTPACookieFromSSOToken() method from the WSSecurityHelper class is deprecated. Use the functionality provided by the getSSOCookieFromSSOToken() method from the WebSecurityHelper class.

Subsequently, the LTPA cookie can be set on an HTTP request header. In this case, the value of the cookie header is the string:

```
ltpaCookie.getName()=ltpaCookie.getValue()
```

For example, if you use org.apache.commons.httpclient.HttpMethod to build your HTTP request, the LTPA cookie can be set as follows:

```
HttpMethod method = .;  // new your HttpMethod based on the
                        // target URL for the web application
if (ltpaCookie != null)
     method.setRequestHeader("Cookie", ltpaCookie.getName()+"="+ltpaCookie.getValue());
```

Note: You should only send LTPA cookies over SSL connections.

Note: You must check whether the LTPA cookie that is returned from calling WSSecurityHelper.getLTPACookieFromSSOToken() in the previous example is not `null` before you issue any `getter` methods. Also, to successfully retrieve a LTPA cookie object, and to ensure an SSO token on the thread of execution, make sure that the user has established a successful authentication with the mid-tier server.

Note: WebSphere Application Server does not ship supporting jars for HTTP programming, such as the Apache httpclient. You must provide your own supporting functions for HTTP programming.

Global single sign-on principal mapping for authentication

You can use the Java Authorization Contract for Containers (JACC) provider for Tivoli Access Manager to manage authentication to enterprise information systems (EIS) such as databases, transaction processing systems, and message queue systems that are located within the WebSphere Application Server security domain. Such authentication is achieved using the global single sign-on (GSO) principal mapper Java Authentication and Authorization Service (JAAS) login module for Java Platform, Enterprise Edition (Java EE) Connector Architecture resources.

With GSO principal mapping, a special-purpose JAAS login module inserts a credential into the subject header. This credential is used by the resource adapter to authenticate to the EIS. The JAAS login module used is configured on a per-connection factory basis. The default principal mapping module retrieves the user name and password information from XML configuration files. The JACC provider for Tivoli Access Manager bypasses the credential that is stored in the Extensible Markup Language (XML) configuration files and uses the Tivoli Access Manager global sign-on (GSO) database instead to provide the authentication information for the EIS security domain.

WebSphere Application Server provides a default principal mapping module that associates user credential information with EIS resources. The default mapping module is defined in the WebSphere Application

Server administrative console on the Application login panel. To access the panel, click **Security > Global security**. Under Java Authentication and Authorization Service, click **Application logins**. The mapping module name is DefaultPrincipalMapping.

The EIS security domain user ID and password are defined under each connection factory by an authDataAlias attribute. The authDataAlias attribute does not contain the user name and password; this attribute contains an alias that refers to a user name and password pair that is defined elsewhere.

The Tivoli Access Manager principal mapping module uses the authDataAlias attribute to determine the GSO resource name and the user name that is required to perform the lookup on the Tivoli Access Manager GSO database. The Tivoli Access Manager Policy Server retrieves the GSO data from the user registry.

Tivoli Access Manager stores authentication information on the Tivoli Access Manager GSO database against a resource and user name pair.

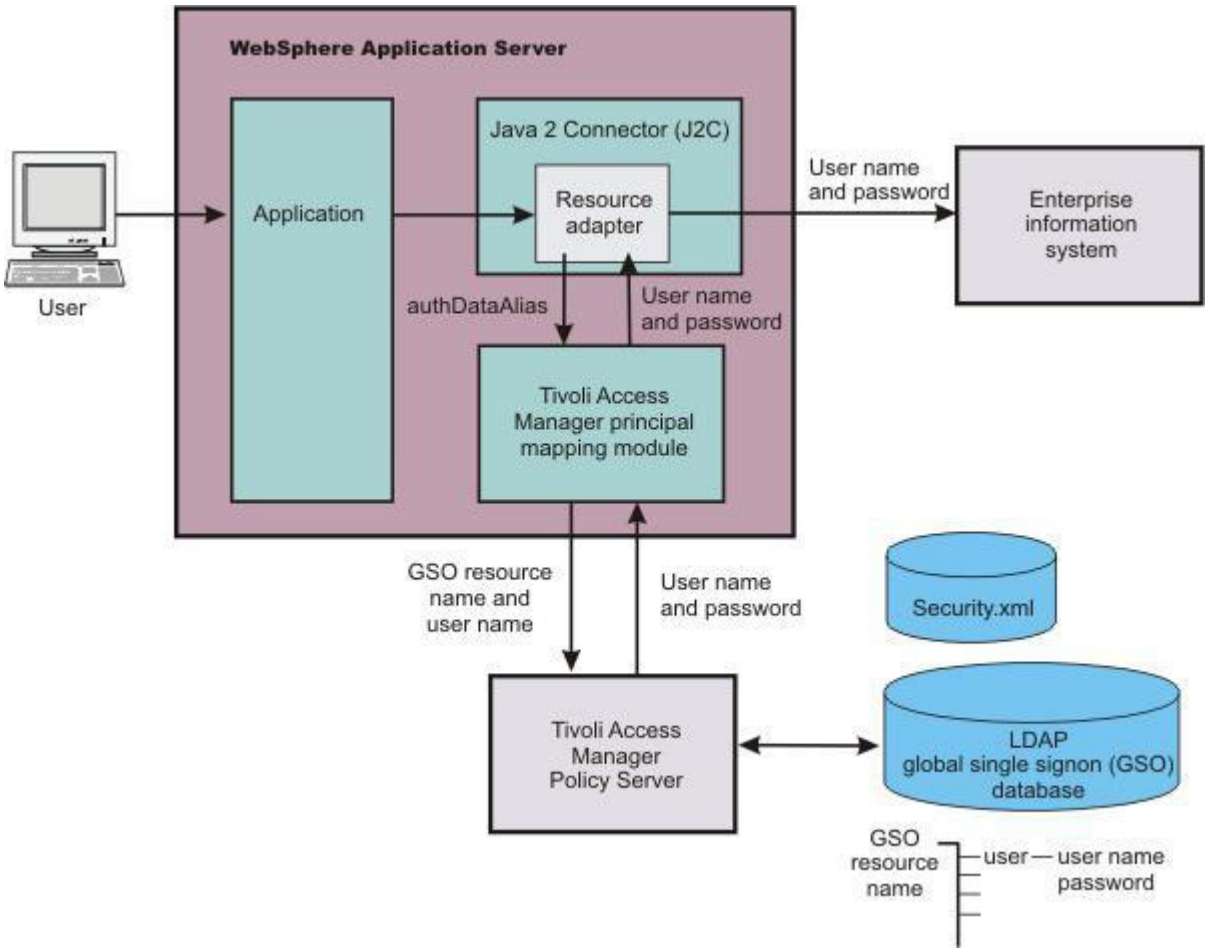


Figure 13. GSO principal mapping architecture

Implementing single sign-on to minimize web user authentications

With single sign-on (SSO) support, web users can authenticate once when accessing web resources across multiple WebSphere Application Servers. Form login mechanisms for web applications require that SSO is enabled. Use this topic to configure single sign-on for the first time.

Before you begin

SSO is supported only when Lightweight Third Party Authentication (LTPA) is the authentication mechanism.

When SSO is enabled, a cookie is created containing the LTPA token and inserted into the HTTP response. When the user accesses other web resources in any other WebSphere Application Server process in the same domain name service (DNS) domain, the cookie is sent in the request. The LTPA token is then extracted from the cookie and validated. If the request is between different cells of WebSphere Application Servers, you must share the LTPA keys and the user registry between the cells for SSO to work. The realm names on each system in the SSO domain are case sensitive and must match identically.

Windows For local OS, the realm name is the domain name if a domain is in use. If a domain is not used, the realm name is the machine name.

Linux AIX HP-UX Solaris The realm name is the same as the host name.

For the Lightweight Directory Access Protocol (LDAP) the realm name is the host:port realm name of the LDAP server. The LTPA authentication mechanism requires that you enable SSO if any of the web applications have form login as the authentication method.

Because single sign-on is a subset of LTPA, it is recommended that you read “Lightweight Third Party Authentication” on page 343 for more information.

When you enable security attribute propagation, the following cookie is always added to the response:

LtpaToken2
LtpaToken2 contains stronger encryption and enables you to add multiple attributes to the token. This token contains the authentication identity and additional information such as the attributes that are used for contacting the original login server and the unique cache key for looking up the Subject when considering more than just the identity in determining uniqueness.

Note: The following cookie is optionally added to the response when the **Interoperability mode** flag is enabled:

LtpaToken
LtpaToken is used for inter-operating with previous releases of WebSphere Application Server. This token contains the authentication identity attribute only.

Note: LtpaToken is generated for releases prior to WebSphere Application Server Version 5.1.1. LtpaToken2 is generated for WebSphere Application Server Version 5.1.1 and beyond.

Table 25. LTPA token types. This table describes the LTPA token types.

Token type	Purpose	How to specify
LtpaToken2 only	This is the default token type. It uses the AES-CBC-PKCS5 padding encryption strength (128-bit key size). This token is stronger than the older LtpaToken used prior to WebSphere Application Server Version 6.02. This is the recommended option when interoperability with older releases is not necessary.	Disable the **Interoperability mode** option in the SSO configuration panel within the administrative console. To access this panel, complete the following steps: 1. Click **Security > Global security**. 2. Under Web security, click **Single sign-on (SSO)**.
LtpaToken and LtpaToken2	Use to interoperate with releases prior to WebSphere Application Server Version 5.1.1. The older LtpaToken cookie is present along with the new LtpaToken2 cookie. Provided the LTPA keys are correctly shared, you should be able to interoperate with any version of WebSphere using this option.	Enable the **Interoperability mode** option in the SSO configuration panel within the administrative console. To access this panel, complete the following steps: 1. Click **Security > Global security**. 2. Under Web security, click **Single sign-on (SSO)**.

About this task

The following steps are required to configure SSO for the first time.

Procedure

1. Open the administrative console.

   Type `http://localhost:port_number/ibm/console` to access the administrative console in a web browser.

   Port 9060 is the default port number for accessing the administrative console. During installation, however, you might have specified a different port number. Use the appropriate port number.
2. Click **Security > Global security**.
3. Under Web security, click **Single sign-on (SSO)**.
4. Click the **Enabled** option if SSO is disabled. After you click the **Enabled** option, make sure that you complete the remaining steps to enable security.
5. Click **Requires SSL** if all of the requests are expected to use HTTPS.
6. Enter the fully qualified domain names in the **Domain name** field where SSO is effective. If you specify domain names, they must be fully qualified. If the domain name is not fully qualified, WebSphere Application Server does not set a domain name value for the LtpaToken cookie and SSO is valid only for the server that created the cookie.

   When you specify multiple domains, you can use the following delimiters: a semicolon (**;**), a space ( ), a comma (**,**), or a pipe (**|**). WebSphere Application Server searches the specified domains in order from left to right. Each domain is compared with the host name of the HTTP request until the first match is located. For example, if you specify `ibm.com®; austin.ibm.com` and a match is found in the ibm.com domain first, WebSphere Application Server does not continue to search for a match in the austin.ibm.com domain. However, if a match is not found in either the ibm.com or austin.ibm.com domains, then WebSphere Application Server does not set a domain for the LtpaToken cookie.

Table 26. Values to configure the Domain name field.

This table describes the values to configure the Domain name field.

Domain name value type	Example	Purpose
Blank		The domain is not set. This causes the browser to set the domain to the request host name. The sign-on is valid on that single host only.
Single domain name	austin.ibm.com	If the request is to a host within the configured domain, the sign-on is valid for all hosts within that domain. Otherwise, it is valid on the request host name only.
UseDomainFromURL	UseDomainFromURL	If the request is to a host within the configured domain, the sign-on is valid for all hosts within that domain. Otherwise, it is valid on the request host name only.
Multiple domain names	austin.ibm.com;raleigh.ibm.com	The sign-on is valid for all hosts within the domain of the request host name.
Multiple domain names and UseDomainFromURL	• austin.ibm.com;raleigh.ibm.com; UseDomainFromURL	The sign-on is valid for all hosts within the domain of the request host name.

If you specify the UseDomainFromURL, WebSphere Application Server sets the SSO domain name value to the domain of the host that makes the request. For example, if an HTTP request comes from server1.raleigh.ibm.com, WebSphere Application Server sets the SSO domain name value to `raleigh.ibm.com` .

Tip: The value, `UseDomainFromURL`, is case insensitive. You can type `usedomainfromurl` to use this value.

For more information, see “Single sign-on settings” on page 422.

7. Optional: Enable the **Interoperability mode** option if you want to support SSO connections in WebSphere Application Server version 5.1.1 or later to interoperate with previous versions of the application server.

   This option sets the old-style LtpaToken token into the response so it can be sent to other servers that work only with this token type. Otherwise, only the LtpaToken2 token is added to the response.

   If performance is a consideration, and you are only connecting to Version 6.1 or later servers that and are not running products that depend on the LtpaToken, do not enable Interoperability mode. When Interoperability mode is not enabled, an LtpaToken is not returned in a response.
8. Optional: Enable the **Web inbound security attribute propagation** option if you want information added during the login at a specific front-end server to propagate to other front-end servers. The SSO token does not contain any sensitive attributes, but does understand where the original login server exists in cases where it needs to contact that server to retrieve serialized information. For more information, see "Security attribute propagation" on page 468.

   Important: If the following statements are true, it is recommended that you disable the **Web inbound security attribute propagation** option for performance reasons:
   - You do not have any specific information added to the Subject during a login that cannot be obtained at a different front-end server.
   - You did not add custom attributes to the PropagationToken token using WSSecurityHelper application programming interfaces (APIs).

   If you find that you are missing custom information in the Subject, re-enable the **Web inbound security attribute propagation** option to see if the information is propagated successfully to other front-end application servers.

   The following two custom properties might help to improve performance when security attribute propagation is enabled:
   - **com.ibm.CSI.propagateFirstCallerOnly**

     The default value of this property is `true`. When this custom property is set to `true` the first caller in the propagation token that stays on the thread is logged when security attribute propagation is enabled. When this property is set to `false`, all of the caller switches are logged, which can affect performance.
   - **com.ibm.CSI.disablePropagationCallerList**

     When this custom property is set to `true` the ability to add a caller or host list in the propagation token is completely disabled. This function is beneficial when the caller or host list in the propagation token is not needed in the environment.
9. Click **OK**.

What to do next

For the changes to take effect, save, stop, and restart all the product servers.

Single sign-on for HTTP requests using SPNEGO web authentication

You can securely negotiate and authenticate HTTP requests for secured resources in WebSphere Application Server by using the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) as the web authentication service for WebSphere Application Server.

Note: In WebSphere Application Server Version 6.1, a trust association interceptor (TAI) that uses the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) to securely negotiate and authenticate HTTP requests for secured resources was introduced. This function was deprecated in WebSphere Application Server Version 7.0. SPNEGO web authentication has taken its place to provide the following enhancements:

- You can configure and enable SPNEGO web authentication and filters on WebSphere Application Server by using the administrative console.
- Dynamic reload of SPNEGO is provided without the need to stop and restart WebSphere Application Server.
- Fallback to an application login method is provided if the SPNEGO web authentication fails.
- SPNEGO can be customized at the WebSphere security domain level. Read about “Multiple security domains” on page 126 for more information.

You can enable either SPNEGO TAI or SPNEGO Web Authentication but not both.

The following sections describe SPNEGO web authentication in more detail:
- “What is SPNEGO?”
- “The benefits of SPNEGO web authentication” on page 378
- “SPNEGO web authentication in a single Kerberos realm” on page 378
- “SPNEGO web authentication in a trusted Kerberos realm” on page 379
- “Support information for SPNEGO web authentication with a Java client using the HTTP protocol” on page 380
- “Support information for SPNEGO web authentication with a browser client” on page 381
- “Setting up SPNEGO as the web authentication mechanism for WebSphere Application Server” on page 381

What is SPNEGO?

SPNEGO is a standard specification defined in The Simple and Protected GSS-API Negotiation Mechanism (IETF RFC 2478).

When WebSphere Application Server global and application security are enabled, and SPNEGO web authentication is enabled, SPNEGO is initialized when processing a first inbound HTTP request. The web authenticator component then interacts with SPNEGO, which is defined and enabled in the security configuration repository. When the filter criteria is met, SPNEGO is responsible for authenticating access to the secured resource that is identified in the HTTP request.

In addition to WebSphere Application Server security runtime services, some external components are required to enable the operation of SPNEGO. These external components include:
- Windows Microsoft Windows Servers with Active Directory domain and associated Kerberos Key Distribution Center (KDC). For information on the supported Microsoft Windows Servers, see the System Requirements for WebSphere Application Server Version 8.5 on Windows.
- A client application, for example, Microsoft .NET, or web service and J2EE client that supports the SPNEGO web authentication mechanism, as defined in IETF RFC 2478. Microsoft Internet Explorer Version 5.5 or later and Mozilla Firefox Version 1.0 are browser examples. Any browser must be configured to use the SPNEGO web authentication mechanism. For more information on performing this configuration, see Configuring the client browser to use SPNEGO.

The authentication of HTTP requests is triggered by the requestor (the client-side), which generates a SPNEGO token. WebSphere Application Server receives this token. Specifically, the SPNEGO web authentication decodes and retrieves the requester's identity from the SPNEGO token. The identity is used to establish a secure context between the requester and the application server.

SPNEGO web authentication is a server-side solution in WebSphere Application Server. Client-side applications are responsible for generating the SPNEGO token for use by SPNEGO web authentication. The requester's identity in the WebSphere Application Server security registry must be identical to the identity that the SPNEGO web authentication retrieves. An identical match does occur when Microsoft Windows Active Directory server is the Lightweight Directory Access Protocol (LDAP) server that is used in

WebSphere Application Server. A custom login module is available as a plug-in to support custom mapping of the identity from the Active Directory to the WebSphere Application Server security registry.

Read about Mapping of a client Kerberos principal name to the WebSphere user registry ID for more information about using this custom login module.

WebSphere Application Server validates the identity against its security registry. If the validation is successful, the client Kerberos ticket and GSS delegation credential are retrieved and placed in the client subject, which then produces a Lightweight Third Party Authentication (LTPA) security token. It then places and returns a cookie to the requester in the HTTP response. Subsequent HTTP requests from this same requester to access additional secured resources in WebSphere Application Server use the LTPA security token previously created to avoid repeated login challenges.

The web administrator has access to the following SPNEGO security components and associated configuration data, as shown in the following figure:

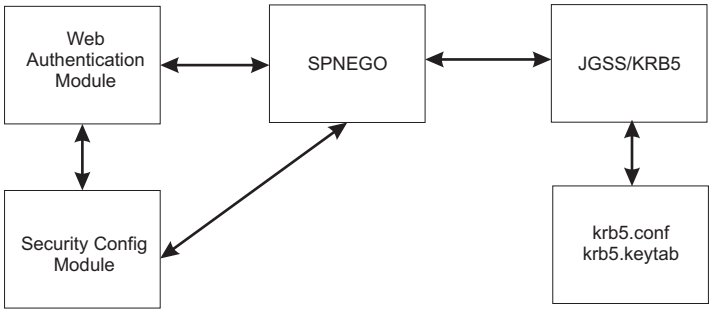


Figure 14. SPNEGO web authentication and security configuration elements

The benefits of SPNEGO web authentication

The benefits of having WebSphere Application Server use SPNEGO as the web authentication service for WebSphere Application Server include the following:

- Windows An integrated single sign-on environment with Microsoft Windows Servers using Active Directory domain is established.
- The cost of administering a large number of ids and passwords is reduced.
- A secure and mutually authenticated transmission of security credentials from the web browser or Microsoft .NET clients is established.
- Interoperability with web services and Microsoft .NET, or web service applications that use SPNEGO authentication at the transport level is achieved.
- With Kerberos authentication support, SPNEGO web authentication can provide an end-to-end SPNEGO to Kerberos solution and preserve the Kerberos credential from the client.

SPNEGO web authentication in a single Kerberos realm

SPNEGO web authentication is supported in a single Kerberos realm. The challenge-response handshake process is shown in the following figure:

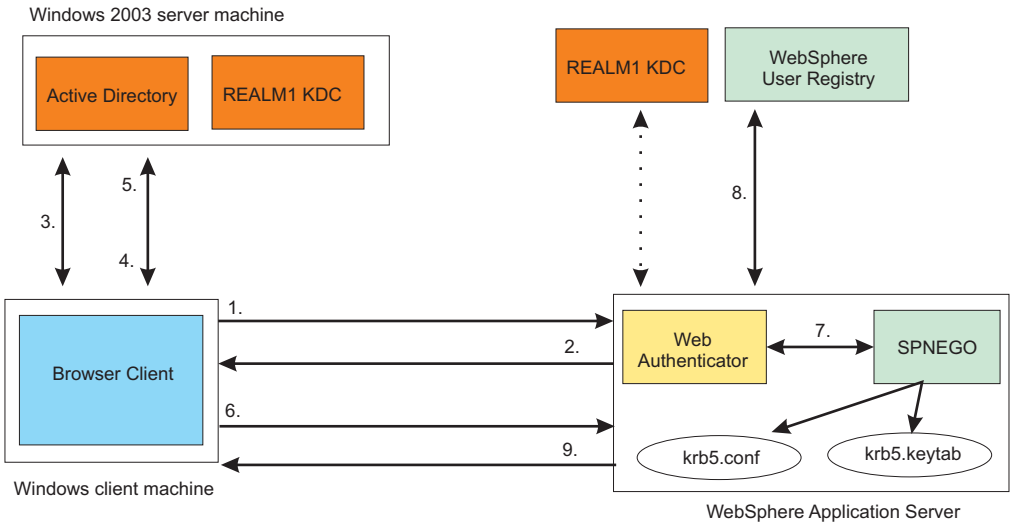


Figure 15. SPNEGO web authentication in a single Kerberos realm

In the previous figure, the following events occur:

1. The client sends an HTTP/Post/Get/Web-Service request to WebSphere Application Server.
2. WebSphere Application Server returns *HTTP 401* Authenticate/Negotiate.
3. The client obtains a Ticket Granting Ticket (TGT).
4. The client requests a Service Ticket (TGS_REQ).
5. The client obtains a Service Ticket (TGS_REP).
6. The client sends HTTP/Post/Get/Web-Service and an authorization SPNEGO token to WebSphere Application Server.
7. WebSphere Application Server validates the SPNEGO token. If the validation is successful, it retrieves the user ID and the GSS delegation credential from the SPNEGO token. Create a KRBAuthnToken with a client Kerberos credential.
8. WebSphere Application Server validates the user ID with the WebSphere user registry and creates an LTPA token.
9. WebSphere Application Server returns *HTTP 200*, content and the LTPA token to the client.

Note: Other clients (for example, web services, .NET and J2EE) that support SPNEGO do not have to follow the challenge-response handshake process as shown previously. Those clients can obtain a ticket-granting ticket (TGT) and a Kerberos service ticket for the target server, create a SPNEGO token, insert it in the HTTP header, and then follow the normal process for creating an HTTP request.

SPNEGO web authentication in a trusted Kerberos realm

SPNEGO web authentication is also supported in a trusted Kerberos realm. The challenge-response handshake process is shown in the following figure:

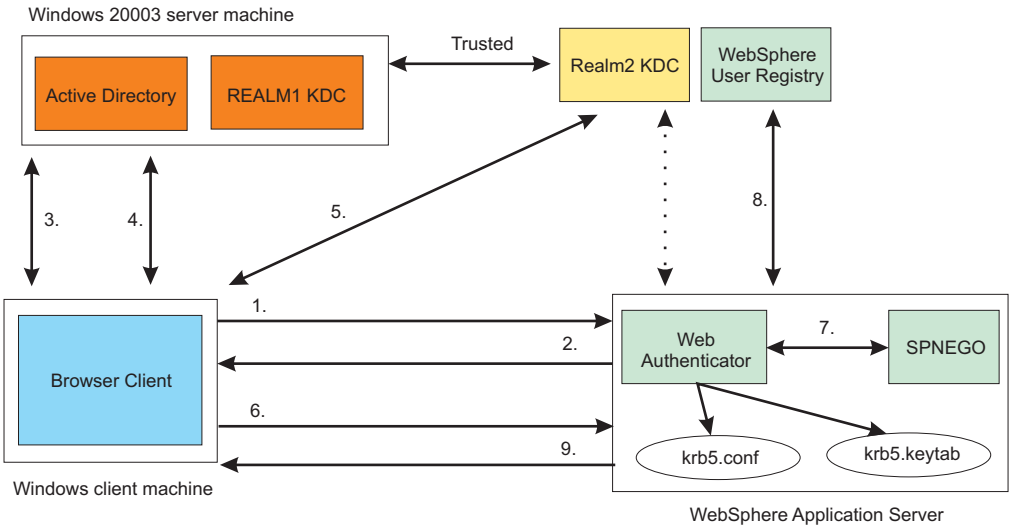


Figure 16. SPNEGO web authentication in a trusted Kerberos realm

In the previous figure, the following events occur:

1. The client sends an HTTP/Post/Get/Web-Service request to WebSphere Application Server.
2. WebSphere Application Server returns *HTTP 401* Authenticate/Negotiate
3. The client obtains a Ticket Granting Ticket (TGT).
4. The client requests a cross realm ticket (TGS_REQ) for *REALM2* from the *REALM1* KDC.
5. The client uses the cross-realm ticket from step 4 to request a Service Ticket from the *REALM2* KDC.
6. The client sends HTTP/Post/Get/Web-Service and an authorization SPNEGO token to WebSphere Application Server.
7. WebSphere Application Server validates the SPNEGO token. If the validation is successful, it retrieves the user ID and the GSS delegation credential from the SPNEGO token. Create a KRBAuthnToken with a client Kerberos credential.
8. WebSphere Application Server validates the user ID with the WebSphere user registry and creates an LTPA token.
9. WebSphere Application Server returns *HTTP 200*, content and the LTPA token to the client.

In the trusted Kerberos realms environment, be aware of the following:

- The Kerberos trusted realm setup must be done on each of the Kerberos KDCs. See your Kerberos Administrator and User's guide for more information about how to set up Kerberos trusted realms.
- The Kerberos client principal name from the SPNEGO token might not exist in the WebSphere user registry; the Kerberos principal mapping to the WebSphere user registry might require it.

  Read about Mapping of a client Kerberos principal name to the WebSphere user registry ID for more information.

Support information for SPNEGO web authentication with a Java client using the HTTP protocol

The following scenarios are supported:

- Domain trust within the same forest
- External domain trust directly between domains within different forests.
- Kerberos realm trust

The following scenarios are not supported:

- Cross-forest trust
- Forest external trust

Support information for SPNEGO web authentication with a browser client

The following scenarios are supported:
- Cross-forest trusts
- Domain trust within the same forest
- Kerberos realm trust

The following scenarios are not supported:
- Forest external trusts
- Domain external trusts

Setting up SPNEGO as the web authentication mechanism for WebSphere Application Server

Before you set up SPNEGO web authentication in the administrative console or by using **wsadmin** commands, you must perform the steps as listed in "Creating a single sign-on for HTTP requests using SPNEGO Web authentication" to set up SPNEGO web authentication for WebSphere Application Server.

Note: SPNEGO web authentication on the server side must be done by the system administrator. The Kerberos keytab file must be protected.

Creating a single sign-on for HTTP requests using SPNEGO Web authentication

Creating single sign-ons for HTTP requests using the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) web authentication for WebSphere Application Server requires the performance of several distinct, yet related functions that when completed, allow HTTP users to log in and authenticate to the Microsoft domain controller only once at their desktop and to receive automatic authentication from the WebSphere Application Server.

Before you begin

Note:

In WebSphere Application Server Version 6.1, a trust association interceptor (TAI) that uses the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) to securely negotiate and authenticate HTTP requests for secured resources was introduced. This function was deprecated in WebSphere Application Server Version 7.0. SPNEGO web authentication has taken its place to provide the following enhancements:
- You can configure and enable SPNEGO web authentication and filters on the WebSphere Application Server server side by using the administrative console.
- Dynamic reload of SPNEGO is provided without the need to stop and restart the WebSphere Application Server server.
- Fallback to an application login method is provided if the SPNEGO web authentication fails.

You can enable either SPNEGO TAI or SPNEGO Web Authentication but not both.

Read about "Single sign-on for HTTP requests using SPNEGO web authentication" on page 376 for a better understanding of what SPNEGO Web Authentication is and how it is supported in this version of WebSphere Application Server.

Before starting this task, complete the following checklist:

- Windows A Microsoft Windows Server running the Active Directory Domain Controller and associated Kerberos Key Distribution Center (KDC). For information on the supported Microsoft Windows Servers, see the System Requirements for WebSphere Application Server Version 8.5 on Windows.
- Windows A Microsoft Windows domain member (client) for example, a browser or Microsoft .NET client, that supports the SPNEGO authentication mechanism, as defined in IETF RFC 2478. Microsoft Internet Explorer Version 5.5 or later and Mozilla Firefox Version 1.0 qualify as such clients.

  Important: A running domain controller and at least one client machine in that domain is required. Using SPNEGO directly from the domain controller is not supported.
- The domain member has users who can log on to the domain. Specifically, you need to have a functioning Microsoft Windows active directory domain that includes:
  - Domain controller
  - Client workstation
  - Users who can login to the client workstation
- A server platform with WebSphere Application Server running and application security enabled.
- Users on the active directory must be able to access WebSphere Application Server protected resources using a native WebSphere Application Server authentication mechanism.
- The domain controller and the host of WebSphere Application Server should have the same local time.
- Ensure the clock on clients, Microsoft Active Directory and WebSphere Application Server are synchronized to within five minutes.
- Be aware that client browsers must be SPNEGO enabled, which you perform on the client application machine (with details explained in procedure 4, "Configure the client application on the client application machine").

About this task

The objective of this machine arrangement is to permit users to successfully access WebSphere Application Server resources without having to authenticate again and thus achieve Microsoft Windows desktop single sign-on capability.

Configuring the members of this environment to establish Microsoft Windows single sign-on involves specific activities that are performed on three distinct machines:

- A Microsoft Windows server running the Active Directory Domain Controller and associated Kerberos Key Distribution Center (KDC).
- A Microsoft Windows domain member (client application), such as a browser or Microsoft .NET client.
- A server platform with WebSphere Application Server running.

Continue with the following steps to create a single sign-on for HTTP requests using SPNEGO Web authentication:

Procedure

1. Create a Kerberos service principal (SPN) and keytab file on your Microsoft domain controller machine You must configure your domain controller machine to create single sign-ons for HTTP requests using the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) web authentication for WebSphere® Application Server. Configure the Microsoft Windows Server running the Active Directory Domain Controller and associated Kerberos Key Distribution Center (KDC).

   Read the Configuring your domain controller machine to create single sign-ons for HTTP requests using SPNEGO article for more information.
2. Create a Kerberos configuration file The IBM implementation of the Java Generic Security Service (JGSS) and KRB5 require a Kerberos configuration file (**krb5.conf** or **krb5.ini**) on each node or Java virtual machine (JVM). In this release of WebSphere Application Server, this configuration file should be placed in the `config/cells/<cell_name>` directory so that all application servers can access this file. If you do not have a Kerberos configuration file, use a **wsadmin** command to create one.

Read the Creating a Kerberos configuration article for more information.

3. Configure and enable SPNEGO web authentication using the administrative console on your WebSphere Application Server machine You can enable and configure the Simple and Protected GSS-API Negotiation (SPNEGO) as the web authenticator for the application server by using the administrative console on the WebSphere Application Server machine.

   Read the Enabling and configuring SPNEGO web authentication using the administrative console article for more information.

4. Configure the client application on the client application machine Client-side applications are responsible for generating the SPNEGO token. You begin this configuration process by configuring your web browser to use SPNEGO authentication.

   Read the Configuring the client browser to use SPNEGO article for more information.

5. Create SPNEGO tokens for J2EE, .NET, Java, web service clients for HTTP requests (optional) You can create a Simple and Protected GSS-API Negotiation (SPNEGO) token for your applications and insert this token into the HTTP headers to authenticate to the WebSphere Application Server.

   Read the Creating SPNEGO tokens for J2EE, .NET, Java, web service clients for HTTP requests article for more information.

Creating a single sign-on for HTTP requests using the SPNEGO TAI (deprecated)

Creating single sign-ons for HTTP requests using the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) trust association interceptor (TAI) for WebSphere Application Server requires the performance of several distinct, yet related functions that when completed, allow HTTP users to log in and authenticate only once at their desktop and receive automatic authentication from the WebSphere Application Server.

Before you begin

Note:

In WebSphere Application Server Version 6.1, a trust association interceptor (TAI) that uses the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) to securely negotiate and authenticate HTTP requests for secured resources was introduced. In WebSphere Application Server 7.0, this function is now deprecated. SPNEGO web authentication has taken its place to provide dynamic reload of the SPNEGO filters and to enable fallback to the application login method.

Before starting this task, complete the following checklist:

- Windows A Microsoft Windows Server running the Active Directory Domain Controller and associated Kerberos Key Distribution Center (KDC). For information on the supported Microsoft Windows Servers, see the System Requirements for WebSphere Application Server Version 8.5 on Windows.
- Windows A Microsoft Windows domain member (client) for example, a browser or Microsoft .NET client, that supports the SPNEGO authentication mechanism, as defined in IETF RFC 2478. Microsoft Internet Explorer Version 5.5 or later and Mozilla Firefox Version 1.0 qualify as such clients.

  Important: A running domain controller and at least one client machine in that domain is required. Trying to use SPNEGO directly from the domain controller is not supported
- The domain member has users who can log on to the domain. Specifically, you need to have a functioning Microsoft Windows active directory domain that includes:
  - Domain controller
  - Client workstation
  - Users who can login to the client workstation
- A server platform with WebSphere Application Server running and application security enabled.
- Users on the active directory must be able to access WebSphere Application Server protected resources using a native WebSphere Application Server authentication mechanism.

- The domain controller and the host of WebSphere Application Server should have the same local time.
- Ensure the clock on clients, Microsoft Active Directory and WebSphere Application Server are synchronized to within five minutes.
- Be aware that client browsers have to be SPNEGO enabled, which you perform on the client application machine (with details explained in step 2 of this task).

About this task

The objective of this machine arrangement is to permit users to successfully access WebSphere Application Server resources without having to reauthenticate and thus achieve Microsoft Windows desktop single sign-on capability.

Configuring the members of this environment to establish Microsoft Windows single sign-on involves specific activities that are performed on three distinct machines:

- Microsoft Windows Server running the Active Directory Domain Controller and associated Kerberos Key Distribution Center (KDC)
- A Microsoft Windows domain member (client application), such as a browser or Microsoft .NET client.
- A server platform with WebSphere Application Server running.

Perform the following steps on the indicated machines to create single sign-on for HTTP requests using SPNEGO

Procedure

1. **Domain Controller Machine** - Configure the Microsoft Windows Server running the Active Directory Domain Controller and associated Kerberos Key Distribution Center (KDC) This configuration activity has the following steps:
   - Create a user account for the WebSphere Application Server in a Microsoft Active Directory. This account will be eventually mapped to the Kerberos service principal name (SPN).
   - On the Microsoft Active Directory machine where the Kerberos key distribution center (KDC) is active, map the user account to the Kerberos service principal name (SPN). This user account represents the WebSphere Application Server as being a Kerberize'd service with the KDC. Use the **setspn** command to map the Kerberos service principal name to a Microsoft user account. The topic, “Creating a Kerberos service principal and keytab file that is used by the WebSphere Application Server SPNEGO TAI (deprecated)” on page 389 has more details about using the **setspn** command.
   - Create the Kerberos keytab file and make it available to WebSphere Application Server. Use the **ktpass** tool to create the Kerberos keytab file (`krb5.keytab`). The topic, “Creating a Kerberos service principal and keytab file that is used by the WebSphere Application Server SPNEGO TAI (deprecated)” on page 389 has more details about using the **ktpass** command. to create the keytab file.

     Note: You make the keytab file available to WebSphere Application Server by copying the `krb5.keytab` file from the Domain Controller (LDAP machine) to the WebSphere Application Server machine. See “Using the ktab command to manage the Kerberos keytab file” on page 393 for more details.

   Important: Your domain controller operations must lead to the following results:
   - A user account is created in the Microsoft Active Directory and mapped to a Kerberos service principal name.
   - A Kerberos keytab file (`krb5.keytab`) is created and made available to the WebSphere Application Server. The Kerberos keytab file contains the Kerberos service principal keys WebSphere Application Server uses to authenticate the user in the Microsoft Active Directory and the Kerberos account.

2. **Client Application Machine** - Configure the client application. Client-side applications are responsible for generating the SPNEGO token for use by the SPNEGO TAI. You begin this configuration process by configuring your web browser to use SPNEGO authentication. See “Configuring the client browser to use SPNEGO TAI (deprecated)” on page 407 for the detailed steps required for your browser.
3. **WebSphere Application Server Machine** - Configure and enable the Application Server and the associated SPNEGO TAI by performing the following tasks:
   - Ensure that LTPA is enabled. See Configuring the Lightweight Third Party Authentication mechanism for more details.
   - Enable the SPNEGO TAI. See “Configuring WebSphere Application Server and enabling the SPNEGO TAI (deprecated)” on page 394 for more details.
   - Create SPNEGO TAI properties using either the wsadmin command task or the administrative console.
     – For using the wsadmin command task, see
       - SpnegoTAICommands group for the AdminTask object (deprecated)
     – For using the administrative console, see “Configuring WebSphere Application Server and enabling the SPNEGO TAI (deprecated)” on page 394 for more details.
   - Configure JVM properties and enable the SPNEGO TAI in Application Server in which it is defined. See “Configuring JVM custom properties, filtering HTTP requests, and enabling SPNEGO TAI in WebSphere Application Server (deprecated)” on page 409 or “Enabling the SPNEGO TAI as JVM custom property using scripting (deprecated)” on page 410 for more details.
   - Install the Kerberos keytab file (created in step 1) on the WebSphere Application Server machine. “Creating a Kerberos service principal and keytab file that is used by the WebSphere Application Server SPNEGO TAI (deprecated)” on page 389 provides the details.
   - Create a basic Kerberos configuration file (krb5.ini or krb5.conf). See The Kerberos configuration file for details.
   - Map the client Kerberos principal name to the WebSphere user registry ID, but only if the WebSphere Application Server does not use Micorsoft Active Directory. See “Mapping Kerberos client principal name to WebSphere user registry ID for SPNEGO TAI (deprecated)” on page 414 for more details.
4. Optional: **Using a remote HTTP server** - To use a remote server, you must complete the following steps, which assume that you have already configured the JVM properties and enabled the SPNEGO TAI in the Application Server in which it is defined (as described in the previous three steps).
   a. Complete the steps in “Creating a Kerberos service principal and keytab file that is used by the WebSphere Application Server SPNEGO TAI (deprecated)” on page 389 for the remote proxy server.
   b. Merge the previous keytab file created in step 1 with the keytab file created in step 4a. See “Using the ktab command to manage the Kerberos keytab file” on page 393 for more information.
   c. Create the SPN for the remote proxy server using the `addSpnegoTAIProperties wsadmin` command task. For more information, see SpnegoTAICommands group for the AdminTask object (deprecated).
   d. Restart the WebSphere Application Server.

Single sign-on for HTTP requests using SPNEGO TAI (deprecated)

WebSphere Application Server provides a trust association interceptor (TAI) that uses the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) to securely negotiate and authenticate HTTP requests for secured resources in WebSphere Application Server.

Note:

In WebSphere Application Server Version 6.1, a trust association interceptor (TAI) that uses the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) to securely negotiate and authenticate HTTP requests for secured resources was introduced. In WebSphere Application

Server 7.0, this function is now deprecated. SPNEGO web authentication has taken its place to provide dynamic reload of the SPNEGO filters and to enable fallback to the application login method.

Read about “Creating a single sign-on for HTTP requests using SPNEGO Web authentication” on page 381 for more information.

SPNEGO is a standard specification defined in The Simple and Protected GSS-API Negotiation Mechanism (IETF RFC 2478).

When WebSphere Application Server administrative security is enabled, the SPNEGO TAI is initialized. While processing inbound HTTP requests, the web authenticator component interacts with the SPNEGO TAI, which is defined and enabled in the security configuration repository. One interceptor is selected and is responsible for authenticating access to the secured resource that is identified in the HTTP request.

Important: The use of TAIs is an optional feature. If no TAI is selected, the authentication process continues normally.

HTTP users log in and authenticate only once at their desktop and are subsequently authenticated (internally) with WebSphere Application Server. The SPNEGO TAI is invisible to the end-user of WebSphere applications. The SPNEGO TAI is only visible to the web administrator who is responsible for ensuring a proper configuration, capacity, and maintenance of the web environment.

In addition to WebSphere Application Server security runtime services, some external components are required to completely enable operation of the SPNEGO TAI. The external components include:

- Windows Microsoft Windows Servers with Active Directory domain and associated Kerberos Key Distribution Center (KDC). For information on the supported Microsoft Windows Servers, see the System Requirements for WebSphere Application Server Version 8.5 on Windows.
- A client application, for example, a browser or Microsoft .NET client, that supports the SPNEGO authentication mechanism, as defined in IETF RFC 2478. Microsoft Internet Explorer Version 5.5 or later and Mozilla Firefox Version 1.0 are browser examples. Any browser needs to be configured to use the SPNEGO mechanism. For more information on performing this configuration, see “Configuring the client browser to use SPNEGO TAI (deprecated)” on page 407.

The authentication of HTTP requests is triggered by the requestor (the client-side), which generates a SPNEGO token. WebSphere Application Server receives this token and validates trust between the requester and WebSphere Application Server. Specifically, the SPNEGO TAI decodes and retrieves the requester's identity from the SPNEGO token. The identity is used to establish a secure context between the requester and the application server.

Remember: The SPNEGO TAI is a server-side solution in WebSphere Application Server. Client-side applications are responsible for generating the SPNEGO token for use by the SPNEGO TAI. The requester's identity in WebSphere Application Server security registry must be identical to that identity the SPNEGO TAI retrieves. An identical match does occur when Microsoft Windows Active Directory server is the Lightweight Directory Access Protocol (LDAP) server that is used in WebSphere Application Server. A custom login module is available as a plug-in to support custom mapping of the identity from the Active Directory to the WebSphere Application Server security registry. See “Mapping Kerberos client principal name to WebSphere user registry ID for SPNEGO TAI (deprecated)” on page 414 for details on using this custom login module.

WebSphere Application Server validates the identity against its security registry and, if the validation is successful, produces a Lightweight Third Party Authentication (LTPA) security token and places and returns a cookie to the requester in the HTTP response. Subsequent HTTP requests from this same requester to access additional secured resources in WebSphere Application Server use the LTPA security token previously created, to avoid repeated login challenges.

The challenge-response handshake process is illustrated in the following graphic:

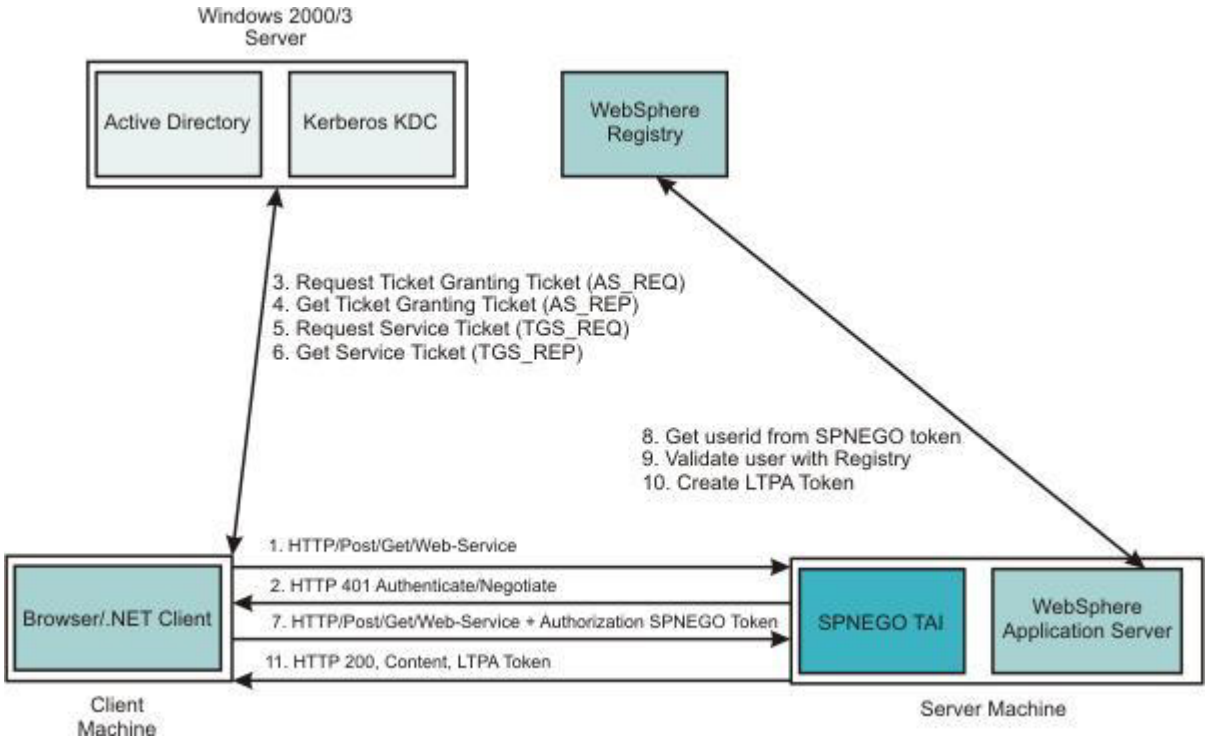


Figure 17. HTTP request processing, WebSphere Application Server - SPNEGO TAI

The SPNEGO TAI can be enabled for all or for selected WebSphere Application Servers in a WebSphere Application Server cell configuration. Also, the behavior of each SPNEGO TAI instance is controlled by custom configuration properties that are used to identify, for example, the criteria used to filter HTTP requests, such as the host name and security realm name used to construct the Kerberos Service Principal Name (SPN). For more information regarding establishing and setting the SPNEGO TAI custom configuration properties, see the following topics:

- Setting up the Kerberos configuration properties. See The Kerberos configuration file.
- Setting or adjusting the SPNEGO TAI custom properties. See “SPNEGO TAI custom properties configuration (deprecated)” on page 403.
- Adjusting the SPNEGO TAI filter settings. See “Configuring JVM custom properties, filtering HTTP requests, and enabling SPNEGO TAI in WebSphere Application Server (deprecated)” on page 409
- Using the custom login module to map the identity from the Active Directory to the WebSphere Application Server registry. See Mapping user Ids from client to server for SPNEGO.
- Setting the major and additional Java virtual machine (JVM) custom properties. See “SPNEGO TAI JVM configuration custom properties (deprecated)” on page 411

The web administrator has access to the following SPNEGO TAI security components and associated configuration data, as illustrated in the following graphic.

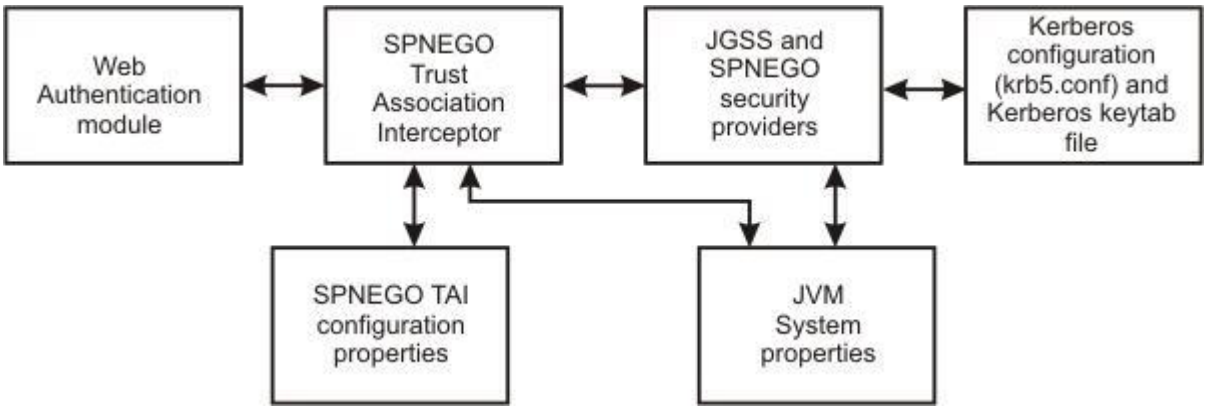


Figure 18. SPNEGO TAI security and configuration elements

- The web authentication module and the Lightweight Third Party Authentication (LTPA) mechanism provide the plug-in runtime framework for trust association interceptors. See Configuring the Lightweight Third Party Authentication mechanism for more detail is configuring the LTPA mechanism for use with the SPNEGO TAI.
- The Java Generic Security Service (JGSS) provider is included in the Java SDK (`jre/lib/ibmjgssprovider.jar`) and used to obtain the Kerberos security context and credentials that are used for authentication. IBM JGSS 1.0 is a Java Generic Security Service Application Programming Interface (GSSAPI) framework with Kerberos V5 as the underlying default security mechanism. GSSAPI is a standardized abstract interface under which can be plugged different security mechanisms based on private-key, public-key and other security technologies. GSSAPI shields secure applications from the complexities and peculiarities of the different underlying security mechanisms. GSSAPI provides identity and message origin authentication, message integrity, and message confidentiality. For more information, see JGSS.
- The Kerberos configuration properties (`krb5.conf` or `krb5.ini`) and Kerberos encryption keys (stored in a Kerberos keytab file) are used to establish secure mutual authentication.

  The Kerberos key table manager (Ktab), which is part of JGSS, allows you to manage the principal names and service keys stored in a local Kerberos keytab file. Principal name and key pairs listed in the Kerberos keytab file allow services running on a host to authenticate themselves to the Kerberos Key Distribution Center (KDC). Before a server can use Kerberos, a Kerberos keytab file must be initialized on the host that runs the server.

  "Using the ktab command to manage the Kerberos keytab file" on page 393 highlights the Kerberos configuration requirements for the SPNEGO TAI as well as the use of Ktab.
- The SPNEGO provider supplies the implementation of the SPNEGO authentication mechanism, located at `/$WAS_HOME/java/jre/lib/ext/ibmspnego.jar`.
- The custom configuration properties control the runtime behavior of the SPNEGO TAI. Configuration operations are performed with the administrative console or scripting facilities. Refer to "SPNEGO TAI custom properties configuration (deprecated)" on page 403 for more information about these custom configuration properties.
- Java virtual machine (JVM) custom properties control diagnostic trace information for problem determination of the JGSS security provider and use of the property reload feature."SPNEGO TAI JVM configuration custom properties (deprecated)" on page 411 describes these JVM custom properties

The benefits of having WebSphere Application Server use the SPNEGO TAI include:

- Windows An integrated single signon environment with Microsoft Windows Servers using Active Directory domain is established.
- The cost of administering a large number of ids and passwords is reduced.
- A secure and mutually authenticated transmission of security credentials from the web browser or Microsoft .NET clients is established.

- Interoperability with web services and Microsoft .NET applications that use SPNEGO authentication at the transport level is achieved.

Using the SPNEGO TAI in your WebSphere Application Server environment requires planning then implementation. See “Single sign-on capability with SPNEGO TAI - checklist (deprecated)” on page 419 in planning for SPNEGO TAI. Implementing the use of the SPNEGO TAI is divided into the following areas of responsibility:

End browser user
The end user must configure the web browser or Microsoft .NET application to issue HTTP requests that are processed by the SPNEGO TAI.

Web administrator
The web administrator is responsible for configuring the SPNEGO TAI of WebSphere Application Server to respond to HTTP requests of the client.

WebSphere Application Server administrator
The WebSphere Application Server administrator is responsible for configuring WebSphere Application Server and the SPNEGO TAI for optimum installation performance.

See “Creating a single sign-on for HTTP requests using the SPNEGO TAI (deprecated)” on page 383 for an explanation of the tasks required to use the SPNEGO TAI and how the responsible party performs these tasks.

Creating a Kerberos service principal and keytab file that is used by the WebSphere Application Server SPNEGO TAI (deprecated)

You perform this configuration task on the Microsoft Active Directory domain controller machine. This task is a necessary part of preparing to process single sign on browser requests to WebSphere Application Server and thee SPNEGO trust association interceptor (TAI).

Before you begin

You need to have a running domain controller and at least one client machine in that domain.

Note:

In WebSphere Application Server Version 6.1, a trust association interceptor (TAI) that uses the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) to securely negotiate and authenticate HTTP requests for secured resources was introduced. In WebSphere Application Server 7.0, this function is now deprecated. SPNEGO web authentication has taken its place to provide dynamic reload of the SPNEGO filters and to enable fallback to the application login method.

About this task

This task is performed on the active directory domain controller machine. Complete the following steps to ensure that the Microsoft Windows Server that is running the active directory domain controller is configured properly to the associated key distribution center (KDC). For information on the supported Microsoft Windows Servers, see the System Requirements for WebSphere Application Server Version 8.5 on Windows.

Procedure

1. Create a user account in the Microsoft Active Directory for the WebSphere Application Server.
   Click **Start->Programs->Administrative Tools->Active Directory Users and Computers**
   Use the name for the WebSphere Application Server. For example, if the Application Server you are running on the WebSphere Application Server machine is called `myappserver.austin.ibm.com`, create a new user in Active Directory called `myappserver`.

Important: Do not select "User must change password at next logon."

Important: Make sure that you do not have the computer name `myappserver` under Computers and Domain Controllers (You check for this condition as illustrated below.). If you already have a computer name `myappserver`, then you need to create a different user account name.

- Goto **Start -> Programs -> Administrative Tools -> Active Directory Users and Computers->Computers**
- Goto **Start -> Programs -> Administrative Tools -> Active Directory Users and Computers->Domain Controllers**

2. Use the **setspn** command to map the Kerberos service principal name, `HTTP/<host name>`, to a Microsoft user account. An example of **setspn** usage is as follows:

```
C:\Program Files\Support Tools>
setspn -A HTTP/myappserver.austin.ibm.com myappserver
```

Note: There may already be some SPNs related to the Microsoft Windows hosts that have been added to the domain. You can display those that exist by using the `setspn -L` command, but you still have to add an HTTP SPN for WebSphere Application Server. For example, `setspn -L myappserver` would list the SPNs.

Important: Make sure that you do not have the same SPNs mapping to more than one Microsoft user account. If you map the same SPN to more than one user account, the web browser client can send a NTLM instead of SPNEGO token to WebSphere Application Server.

More information about the **setspn** command can be found here, Windows 2003 Technical Reference (setspn command)

3. Create the Kerberos keytab file and make it available to WebSphere Application Server. Use the ktpass command to create the Kerberos keytab file (krb5.keytab).

   Use the **ktpass** tool from the Windows Server toolkit to create the Kerberos keytab file for the service principal name (SPN). Use the latest version of the **ktpass** tool that matches the Windows server level that you are using. For example, use the Windows 2003 version of the tool for a Windows 2003 server.

   To determine the appropriate parameter values for the **ktpass** tool, run the `ktpass -?` command from the command line. This command lists whether the **ktpass** tool, which corresponds to the particular operating system, uses the `-crypto RC4-HMAC` or `-crypto RC4-HMAC-NT` parameter value. To avoid warning messages from the toolkit, you must specify the `-ptype KRB5_NT_PRINCIPAL` parameter value.

   The Windows 2003 server version of the **ktpass** tool supports the encryption type, RC4-HMAC, and Single data encryption standard (DES). For more information about the **ktpass** tool, see Windows 2003 Technical Reference (Kerberos keytab file and ktpass command).

   The following code shows the functions that are available when you enter `ktpass -?` command on the command line. This information might be different depending on the version of the toolkit that you are using.

```
C:\Program Files\Support Tools>ktpass -?
Command line options:

--------------------most useful args
[- /]          out : Keytab to produce
[- /]        princ : Principal name (user@REALM)
[- /]         pass : password to use
                     use "*" to prompt for password.
[- +]      rndPass : ... or use +rndPass to generate a random password
[- /]      minPass : minimum length for random password (def:15)
[- /]      maxPass : maximum length for random password (def:256)
--------------------less useful stuff
[- /]      mapuser : map princ (above) to this user account (default:
don't)
[- /]        mapOp : how to set the mapping attribute (default: add it)
[- /]        mapOp :  is one of:
[- /]        mapOp :        add : add value (default)
[- /]        mapOp :        set : set value
[- +]      DesOnly : Set account for des-only encryption (default:don't)
```

```
[- /]           in : Keytab to read/digest
--------------------options for key generation
[- /]       crypto : Cryptosystem to use
[- /]       crypto :  is one of:
[- /]       crypto : DES-CBC-CRC : for compatibility
[- /]       crypto : DES-CBC-MD5 : for compatibliity
[- /]       crypto : RC4-HMAC-NT : default 128-bit encryption
[- /]        ptype : principal type in question
[- /]        ptype :  is one of:
[- /]        ptype : KRB5_NT_PRINCIPAL : The general ptype-- recommended
[- /]        ptype : KRB5_NT_SRV_INST : user service instance
[- /]        ptype : KRB5_NT_SRV_HST : host service instance
[- /]         kvno : Override Key Version Number
                     Default: query DC for kvno.  Use /kvno 1 for Win2K
compat.
[- +]       Answer : +Answer answers YES to prompts.  -Answer answers
NO.
[- /]       Target : Which DC to use.  Default:detect
---------------------options for trust attributes (Windows Server 2003
Sp1 Only
[- /] MitRealmName : MIT Realm which we want to enable RC4 trust on.
[- /]  TrustEncryp : Trust Encryption to use; DES is default
[- /]  TrustEncryp :  is one of:
[- /]  TrustEncryp :        RC4 : RC4 Realm Trusts (default)
[- /]  TrustEncryp :        DES : go back to DES
```

Important: Do not use the -pass switch on the **ktpass** command to reset a password for a Microsoft Windows server account.

See Windows 2003 Technical Reference (Kerberos keytab file and ktpass command) for more information. You must use the -mapUser option with **ktpass** command to enable the KDC to create an encryption key. Otherwise, when the SPENGO token is received, it fails the validation process and the application server challenges the user for a user name and password.

Depending on the encryption type, you use the **ktpass** tool in one of the following ways to create the Kerberos keytab file. The following section shows the different types of encryption that are used by the ktpass tool. It is important that you run the ktpass -? command to determine which -crypto parameter value is expected by the particular toolkit in your Microsoft Windows environment.

- For a single DES encryption type

  From a command prompt, run the **ktpass** command:

```
ktpass -out c:\temp\myappserver.keytab
-princ HTTP/myappserver.austin.ibm.com@WSSEC.AUSTIN.IBM.COM
-mapUser myappserv
-mapOp set
-pass was1edu
-crypto DES-CBC-MD5
-pType KRB5_NT_PRINCIPAL
+DesOnly
```

Table 27. Using ktpass for a single DES encryption type.

This table describes how to use ktpass for a single DES encryption type.

Option	Explanation
-out c:\temp\myappserver.keytab	The key is written to this output file.
-princ HTTP/ myappserver.austin.ibm.com@WSSEC.AUSTIN.IBM.COM	The concatenation of the user logon name, and the realm must be in uppercase.
-mapUser	The key is mapped to the user, myappserver.
-mapOp	This option sets the mapping.
-pass was1edu	This option is the password for the user ID.
-crypto DES-CBC-MD5	This option uses the single DES encryption type.
-pType KRB5_NT_PRINCIPAL	This option specifies the KRB5_NT_PRINCIPAL principal value. Specify this option to avoid toolkit warning messages.
+DesOnly	This option generates only DES encryptions.

- For the RC4-HMAC encryption type

Important: RC4-HMAC encryption is only supported when using a Windows 2003 Server as KDC. From a command prompt, run the **ktpass** command.

```
ktpass -out c:\temp\myappserver.keytab
-princ HTTP/myappserver.austin.ibm.com@WSSEC.AUSTIN.IBM.COM
-mapUser myappserver
-mapOp set
–pass was1edu
-crypto RC4-HMAC
-pType KRB5_NT_PRINCIPAL
```

Table 28. Using ktpass for the RC4-HMAC encryption type.

This table identifies and describes the ktpass options for RC4-HMAC encryption

Option	Explanation
`-out c:\temp\myappserver.keytab`	The key is written to this output file.
`-princ HTTP/ myappserver.austin.ibm.com@WSSEC.AUSTIN.IBM.COM`	The concatenation of the user logon name, and the realm must be in uppercase.
`-mapUser`	The key is mapped to the user, `myappserver`.
`-mapOp`	This option sets the mapping.
`-pass was1edu`	This option is the password for the user ID.
`-crypto RC4-HMAC`	This option chooses the RC4-HMAC encryption type.
`-pType KRB5_NT_PRINCIPAL`	This option specifies the `KRB5_NT_PRINCIPAL` principal value. Specify this option to avoid toolkit warning messages.

- For the RC4-HMAC-NT encryption type

  From a command prompt, run the **ktpass** command.

```
ktpass -out c:\temp\myappserver.keytab
-princ HTTP/myappserver.austin.ibm.com@WSSEC.AUSTIN.IBM.COM
-mapUser myappserver
-mapOp set
-pass was1edu
-crypto RC4-HMAC-NT
-pType KRB5_NT_PRINCIPAL
```

Table 29. Using ktpass for the RC4-HMAC encryption type. This table describes the use of ktpass for RC4-HMAC encryption types.

Option	Explanation
`-out c:\temp\myappserver.keytab`	The key is written to this output file.
`-princ HTTP/ myappserver.austin.ibm.com@WSSEC.AUSTIN.IBM.COM`	The concatenation of the user logon name, and the realm must be in uppercase.
`-mapUser`	The key is mapped to the user, `myappserver`.
`-mapOp`	This option sets the mapping.
`-pass was1edu`	This option is the password for the user ID.
`-crypto RC4-HMAC-NT`	This option chooses the RC4-HMAC-NT encryption type.
`-pType KRB5_NT_PRINCIPAL`	This option specifies the `KRB5_NT_PRINCIPAL` principal value. Specify this option to avoid toolkit warning messages.

The Kerberos keytab file is created for use with the SPNEGO TAI.

Note: A Kerberos keytab configuration file contains a list of keys that are analogous to user passwords. It is important for hosts to protect their Kerberos keytab files by storing them on the local disk, which makes them readable only be authorized users.

You make the keytab file available to WebSphere Application Server by copying the `krb5.keytab` file from the Domain Controller (LDAP machine) to the WebSphere Application Server machine.

```
ftp> bin
ftp> put c:\temp\KRB5_NT_SEV_HST\krb5.keytab
```

Results

Your active directory domain controller is properly configured to process single sign on requests to WebSphere Application Server and the SPNEGO TAI

Using the ktab command to manage the Kerberos keytab file:

The Kerberos key table manager command (Ktab) allows the product administrator to manage the Kerberos service principal names and keys stored in a local Kerberos keytab file. With the IBM Software Development Kit (SDK) or Sun Java Development Kit (JDK) 1.6 or later, you can use the ktab command to merge two Kerberos keytab files.

Solaris To merge the ktab files, you must install Java Development Kit (JDK) Version 1.6 SR3 cumulative fix, which upgrades the JDK to Version 1.6.0_07.

HP-UX To merge the ktab files, you must install Software Development Kit (SDK) Version 1.6 SR3 cumulative fix, which upgrades the JDK to Version 1.6.0.02.

AIX Windows Linux To merge the ktab files, you must install Java Development Kit (JDK) Version 1.6 SR3 cumulative fix, which upgrades the SDK to Version 1.6.0 Java Technology Edition SR3.

Kerberos service principal (SPN) name and keys listed in the Kerberos keytab file allow services running on the host to validate the incoming Kerberos or SPNEGO token request. Prior to configuring Kerberos or SPNEGO web authentication, the WebSphere Application Server administrator must setup a Kerberos keytab file on the host that is running WebSphere Application Server.

Note:

In WebSphere Application Server Version 6.1, a trust association interceptor (TAI) that uses the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) to securely negotiate and authenticate HTTP requests for secured resources was introduced. In WebSphere Application Server Version 7.0, this function is now deprecated.

SPNEGO web authentication has taken its place to provide the following enhancements:

- Configure and enable SPNEGO Web Authentication and filters on the WebSphere Application Server side by using the administrative console.
- Provide dynamic reload of SPNEGO without having to stop and restart the WebSphere Application Server.
- Provide fallback to an application login method if the SPNEGO web authentication fails.

Important:

- It is important to protect the keytab files and make them readable only by authorized product users.
- Any updates to the Kerberos keytab file using Ktab do not affect the Kerberos database. If you change the keys in the Kerberos keytab file, you must also make the corresponding changes to the Kerberos database.

The syntax of Ktab is illustrated later in this section by using Ktab with the `-help` operand.

```
$ ktab -help

Usage: java com.ibm.security.krb5.internal.tools.Ktab [options]
Available options:
-l                            list the keytab name and entries
-a <principal_name> [password]  add an entry to the keytab
```

```
-d <principal_name>             delete an entry from the keytab
-k <keytab_name>                specify keytab name and path with FILE: prefix
-m <source_keytab_name> <destination_keytab_name>      specify merging source keytab file name and destination keytab file name
```

Following is an example of how Ktab is used to merge the krb5Host1.keytab file to the krb5.keytab file:

```
[root@wssecjibe bin]# ./ktab -m /etc/krb5Host1.keytab /etc/krb5.keytab
Merging keytab files:   source=krb5Host1.keytab   destination=krb5.keytab
Done!
[root@wssecjibe bin]# ls /etc/krb5.*
/etc/krb5Host1.keytab/etc/krb5.keytab
/etc/krb5.keytab
```

Following is an example of how Ktab is used on a LINUX platform to add new principal names to the Kerberos keytab file, where ot56prod is the password for the Kerberos principal name:

```
[root@wssecjibe bin]# ./ktab -a
HTTP/wssecjibe.austin.ibm.com@WSSEC.AUSTIN.IBM.COM ot56prod -k /etc/krb5.keytab
Done!
Service key for principal HTTP/wssecjibe.austin.ibm.com@WSSEC.AUSTIN.IBM.COM saved
```

Following is an example of how Ktab is used on a Linux platform to list Kerberos keytab file content.

```
[root@wssecjibe bin]# ./ktab

        KVNO    Principal
        ----    ---------

        1       HTTP/wssecjibe.austin.ibm.com@WSSEC.AUSTIN.IBM.COM

[root@wssecjibe bin]# ls /etc/krb5.*
/etc/krb5.conf
/etc/krb5.keytab
```

Tip: You can run the ktab command from the *install_root*/java/jre/bin directory.

Configuring WebSphere Application Server and enabling the SPNEGO TAI (deprecated)

Performing this task helps you, as web administrator, to ensure that WebSphere Application Server is properly configured to enable the operation of the Simple and Protected GSS-API Negotiation (SPNEGO) trust association interceptor (TAI).

Before you begin

You need to know how to use the WebSphere Application Server administrative console to manage the security configuration and have the proper authority to modify the security configuration of the application server.

Note:

> In WebSphere Application Server Version 6.1, a trust association interceptor (TAI) that uses the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) to securely negotiate and authenticate HTTP requests for secured resources was introduced. In WebSphere Application Server 7.0, this function is now deprecated. SPNEGO web authentication has taken its place to provide dynamic reload of the SPNEGO filters and to enable fallback to the application login method.

About this task

Complete the following steps to enable the operation of the SPNEGO TAI.

Procedure

1. Log on to the WebSphere Application Server administrative console.
2. Click **Security > Global security**.
3. Expand **Web security** and click **Trust association**.
4. Under the General Properties heading, select the **Enable trust association** check box, then click **Interceptors**.
5. Select the SPNEGO TAI in the list of interceptors.
6. Then click **Custom properties**.
7. Click **New** and then fill in the **Name** and **Value** fields. Click **OK**. Repeat this step for each custom property that you want to apply to the SPNEGO TAI. See “SPNEGO TAI custom properties configuration (deprecated)” on page 403 for a complete list of SPNEGO TAI custom properties.

   Note: It is recommended that you use the **wsadmin** utility to manage the SPNEGO TAI properties. You can add, modify, and delete SPNEGO TAI properties as well as display them using wsadmin. See “Adding SPNEGO TAI properties using the wsadmin utility (deprecated)” on page 396 to add, “Modifying SPNEGO TAI properties using the wsadmin utility (deprecated)” on page 399 to modify, and “Deleting SPNEGO TAI properties using the wsadmin utility (deprecated)” on page 401 to delete SPNEGO TAI properties.
8. After you finish defining your custom properties, click **Save** to store the updated SPNEGO TAI configuration.
9. Optional: If an alias for a connecting host name is added dynamically after the application server is started, you need to configure the alias. Refer to the “Using an alias host name for SPNEGO TAI or SPENGO web authentication using the administrative console (deprecated)” topic.

Results

Your SPNEGO TAI configuration is now configured for WebSphere Application Server.

Using an alias host name for SPNEGO TAI or SPENGO web authentication using the administrative console (deprecated):

When you use the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) trust association interceptor (TAI) for authentication, and you would like to use alias host name as the host name for the application server, you must configure a custom property to resolve the alias host name to the actual hostname for SPNEGO single sign-on. Then, you can dynamically add or modify an alias name in the DNS without changing the application server's configuration. If you enable this custom property you will no longer need to set alias host names through the SPNEGO configuration.

Before you begin

You must have completed the steps as described in “Creating a single sign-on for HTTP requests using the SPNEGO TAI (deprecated)” on page 383 and “Configuring WebSphere Application Server and enabling the SPNEGO TAI (deprecated)” on page 394 before these settings will have an effect. This configuration requires a working SPNEGO-TAI single sign-on environment.

About this task

The application server will perform a DNS lookup as an HTTP request comes in, and if the alias host name is resolved as a host name that is already configured for SPNEGO single sign-on, the application server will continue to process it. It is usually not required to add alias hostname to a SPNEGO account.

Procedure

1. Define the actual host name for the com.ibm.ws.security.spnego.SPNx.hostName variable.

a. From administration console, click **Global security** > **Web and SIP security** > **Trust association** > **Interceptors** > **com.ibm.ws.security.spnego.TrustAssociationInterceptorImpl** > **Custom Properties**

b. Add or modify the com.ibm.ws.security.spnego.SPNx.hostName variable. For example:

Name com.ibm.ws.security.spnego.SPNx.hostName

Value *real_host_name*

This custom property specifies the actual host name to which the application server can resolve an alias host name for SPNEGO single sign-on. You can then dynamically add or modify an alias name in the DNS without changing the configuration for the application server.

You can optionally define the alias host name, but you are only required to define the real host name. The application server resolves the alias host name to real host name as the HTTP request is received.

2. Turn on the Canonical support flag.

a. From administration console, click **Global security** > **Custom properties**

b. Add or modify the com.ibm.websphere.security.krb.canonical_host variable and set it to "true".

Name com.ibm.websphere.security.krb.canonical_host

Value true

This custom property specifies whether the application server uses the canonical form of the URL/HTTP host name in authenticating a client. If you set this custom property to `false`, a Kerberos ticket can contain a host name that differs from the HTTP host name header and the application server might issue the following message:

```
CWSPN0011E: An invalid SPNEGO token has been encountered while authenticating a HttpServletRequest
```

If you set this custom property to `true`, you can avoid this error message and allow the application server to authenticate using the canonical form of the URL/HTTP host name.

3. Configure the browser. On the browser for the client machine, the alias host name needs to be configured as a trusted host.

- For Internet Explorer:
  a. Select **Tools** > **Internet options**.
  b. Select the Security tab.
  c. Click **Local intranet** > **Sites** > **Advanced**
  d. Add the alias host name in this panel.
- For Mozilla Firefox:
  a. Type **About:config** in the address bar and press ENTER to access configuration options.
  b. Locate the **network.negotiate-auth.trusted-uris** preference name, right-click on the preference, and select **Modify**. If you do not have this preference, right-click within the panel, and select **New** > **string**.
  c. Add alias host names in the text box, separating host names with a comma.

4. Ensure that the real host name is added to the keytab file.

config: You can configure the keytab file in two ways:

- If com.ibm.websphere.security.krb.canonical_host is set to "true", the application server expects the real host name to be in the keytab files. Aliases are not necessary.
- If com.ibm.websphere.security.krb.canonical_host is set to false and aliases are defined, aliases need to be present in the keytab file.

Adding SPNEGO TAI properties using the wsadmin utility (deprecated):

You use the wsadmin utility to add properties for the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) trust association interceptor (TAI) in the security configuration for WebSphere Application Server.

About this task

Note:

In WebSphere Application Server Version 6.1, a trust association interceptor (TAI) that uses the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) to securely negotiate and authenticate HTTP requests for secured resources was introduced. In WebSphere Application Server 7.0, this function is now deprecated. SPNEGO web authentication has taken its place to provide dynamic reload of the SPNEGO filters and to enable fallback to the application login method.

Use the wsadmin utility to configure the SPNEGO TAI for WebSphere Application Server:

Procedure

1. Start WebSphere Application Server.
2. Start the command-line utility by running the **wsadmin** command from the *app_server_root*/bin directory.
3. At the **wsadmin** prompt, enter the following command:

   ```
   $AdminTask addSpnegoTAIProperties
   ```

   You can use the following parameters with this command:

Option	Description
<spnId>	This parameter is optional. It is the SPN identifier for the group of custom properties that are to be defined with this command. If you do not specify this parameter, an unused SPN identifier is assigned.
<host>	This parameter is required. It specifies the host name portion in the SPN used by the SPNEGO TAI to establish a Kerberos secure context.
<filter>	This parameter is optional. It defines the filtering criteria used by the class specified with the above attribute. If you do not specify this parameter, all HTTP requests are subject to SPNEGO authentication.
<filterClass>	This parameter is optional. It specifies the name of the Java class used by the SPNEGO TAI to select which HTTP requests will be subject to SPNEGO authentication. If you do not specify this paramter, the default filter class, com.ibm.ws.security.spnego.HTTPHeaderFilter, is used.

Option	Description
<noSpnegoPage>	This parameter is optional. It specifies the URL of a resource that contains the content the SPNEGO TAI will include in the HTTP response to be displayed by the (browser) client application if it does not support SPNEGO authentication. If you do not specify the noSpnegoPage paramter then the default is used: `"<html><head><title>SPNEGO authentication is not supported. </title></head>" + "<body>SPNEGO authentication is not supported on this client. </body></html>";`
<ntlmTokenPage>	This parameter is optional. It specifies the URL of a resource that contains the content the SPNEGO TAI will include in the HTTP response that is to be displayed by the (browser) client application when the SPNEGO token received by the interceptor (after the challenge-response handshake) contains a NT LAN manager (NTLM) token instead of the expected SPNEGO token. If you do not specify the ntlmTokenPage parameter then the default is used: `"<html><head><title>An NTLM Token was received.</title></head>" + "<body>Your browser configuration is correct, but you have not logged into a supported Windows Domain." + "<p>Please login to the application using the normal login page.</html>";`
<trimUserName>	This parameter is optional. It specifies whetheror not the SPNEGO TAI is to remove the suffix of the principal user name, starting from the “@” that precedes the Kerberos realm name. If this parameter is set to `true`, the suffix of the principal user name is removed. If this paramter is set to `false`, the suffix of the principal name is retained. The default value used is `true`.

Results

SPNEGO TAI properties have been added for this WebSphere Application Server.

Example

Example 1

The following example configures the SPNEGO TAI to intercept HTTP requests that contain `IE 6` in the user agent request header. The SPNEGO TAI uses the SPN of HTTP/ myhost.ibm.com@<default_realm> to authenticate the request originator.

```
$AdminTask addSpnegoTAIProperties -host myhost.ibm.com -filter user-agent%=IE 6
```

Example 2

The following is an example of adding SPNEGOTAIProperties for SPN1 to use the default filterClass and to intercept all requests for the host, central01.austin.ibm.com.

```
wsadmin>$AdminTask addSpnegoTAIProperties -interactive
Add SPNEGO TAI properties

Add SPNEGO TAI configuration properties.

*Host name in Service Principal Name (host): central01.austin.ibm.com
Service Principal Name identifier (spnId): 1
HTTP header filter rule (filter):
Name of class used to filter HTTP requests (filterClass):
SPNEGO not supported browser response (noSpnegoPage):
NTLM Token received browser response (ntlmTokenPage):
Trim User Name browser response (trimUserName):

Add SPNEGO TAI properties

F (Finish)
C (Cancel)

Select [F, C]: [F] f
WASX7278I: Generated command line: $AdminTask addSpnegoTAIProperties {-host central01.austin.ibm.com}
com.ibm.ws.security.spnego.SPN1.hostName=central01.austin.ibm.com
wsadmin>
```

Modifying SPNEGO TAI properties using the wsadmin utility (deprecated):

You use the wsadmin utility to modify the properties in the configuration of the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) trust association interceptor (TAI) for WebSphere Application Server.

About this task

Note:

> In WebSphere Application Server Version 6.1, a trust association interceptor (TAI) that uses the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) to securely negotiate and authenticate HTTP requests for secured resources was introduced. In WebSphere Application Server 7.0, this function is now deprecated. SPNEGO web authentication has taken its place to provide dynamic reload of the SPNEGO filters and to enable fallback to the application login method.

You use the wsadmin utility to configure the SPNEGO TAI for WebSphere Application Server:

Procedure

1. Start WebSphere Application Server.
2. Start the command-line utility by running the **wsadmin** command from the *app_server_root*/bin directory.
3. At the **wsadmin** prompt, enter the following command:

   ```
   $AdminTask modifySpnegoTAIProperties
   ```

   You can use the following parameters with this command:

Option	Description
<spnId>	This parameter is required. It is the SPN identifier for the group of custom properties that are to be defined with this command.
<host>	This parameter is optional. It specifies the host name portion in the SPN used by the SPNEGO TAI to establish a Kerberos secure context.

Option	Description
<filter>	This parameter is optional. It defines the filtering criteria used by the class specified with the above attribute.
<filterClass>	This parameter is optional. It specifies the name of the Java class used by the SPNEGO TAI to select which HTTP requests will be subject to SPNEGO authentication. If no class is specified, all HTTP requests will be subject to SPNEGO authentication.
<noSpnegoPage>	This parameter is optional. It specifies the URL of a resource that contains the content the SPNEGO TAI will include in the HTTP response to be displayed by the (browser) client application if it does not support SPNEGO authentication. If you do not specify the noSpnegoPage attribute then the default is used: `"<html><head><title>SPNEGO authentication is not supported. </title></head>" + "<body>SPNEGO authentication is not supported on this client. </body></html>";`
<ntlmTokenPage>	This parameter is optional. The ntlmTokenPage parameter specifies the URL of a resource that contains the content the SPNEGO TAI will include in the HTTP response, which will be displayed by the (browser) client application. The (browser) client application displays this HTTP response when the browser client sends a NT LAN manager (NTLM) token instead of the expected SPNEGO token during the challange-response handshake. If you do not specify the ntlmTokenPage attribute then the default is used: `"<html><head><title>An NTLM Token was received.</title></head>" + "<body>Your browser configuration is correct, but you have not logged into a supported Windows Domain." + "<p>Please login to the application using the normal login page.</html>";`
<trimUserName>	This parameter is optional. It specifies whether (`true`) or not (`false`) the SPNEGO TAI is to remove the suffix of the principal user name, starting from the "@" that precedes the Kerberos realm name. If this attribute is set to `true`, the suffix of the principal user name is removed. If this attribute is set to `false`, the suffix of the principal name is retained. The default value used is `true`.

Results

SPNEGO TAI properties are modified for this WebSphere Application Server.

Example

Example 1

The following example configures the SPNEGO TAI to intercept HTTP requests that contain `IE 6` in

the user agent request header. The SPNEGO TAI uses the SPN of HTTP/ myhost.ibm.com@<default_realm> to authenticate the request originator. Then the example modifies the value of the filter custom property that was defined and changes it from `user-agent%=IE 6` to `host==myhost.company.com`.

```
$AdminTask addSpnegoTAIProperties -host myhost.ibm.com -filter user-agent%=IE 6
$AdminTask modifySpnegoTAIProperties -spnId 1 -filter host==myhost.company.com
```

Example 2

This is an example of modifying the SPNEGO TAI for SPN1 properties to add a filter for host central01.austin.ibm.com.

```
wsadmin>$AdminTask modifySpnegoTAIProperties -interactive
Modify SPNEGO TAI properties

Modify SPNEGO TAI configuration properties

*Service Principal Name identifier (spnId): 1
Host name in Service Principal Name (host): central01.austin.ibm.com
HTTP header filter rule (filter): request-url!=noSPNEGO;request-url%=snoop
Name of class used to filter HTTP requests (filterClass):
SPNEGO not supported browser response (noSpnegoPage):
NTLM Token received browser response (ntlmTokenPage):
Trim User Name browser response (trimUserName):

Modify SPNEGO TAI properties

F (Finish)
C (Cancel)

Select [F, C]: [F] f
WASX7278I: Generated command line: $AdminTask modifySpnegoTAIProperties {-spnId
1 -host w2003secdev.austin.ibm.com -filter request-url!=noSPNEGO;request-url%=sn
oop}
com.ibm.ws.security.spnego.SPN1.filter=request-url!=noSPNEGO;request-url%=snoop
com.ibm.ws.security.spnego.SPN1.hostName=central01.austin.ibm.com
wsadmin>
```

Deleting SPNEGO TAI properties using the wsadmin utility (deprecated):

You use the wsadmin utility to delete properties in the configuration of the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) trust association interceptor (TAI) for WebSphere Application Server.

About this task

Note:

> In WebSphere Application Server Version 6.1, a trust association interceptor (TAI) that uses the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) to securely negotiate and authenticate HTTP requests for secured resources was introduced. In WebSphere Application Server 7.0, this function is now deprecated. SPNEGO web authentication has taken its place to provide dynamic reload of the SPNEGO filters and to enable fallback to the application login method.

You use the wsadmin utility to configure the SPNEGO TAI for WebSphere Application Server:

Procedure

1. Start WebSphere Application Server.
2. Start the command-line utility by running the **wsadmin** command from the *app_server_root*/bin directory.

3. At the **wsadmin** prompt, enter the following command:
   ```
   $AdminTask deleteSpnegoTAIProperties
   ```

   You can use the following parameters with this command:

Option	Description
<spnId>	This is an optional parameter. It is the SPN identifier for the group of custom properties that are to be deleted with this command. If you do not specify this parameter, all SPNEGO TAI custom properties are deleted.

Results

SPNEGO TAI properties are deleted for this WebSphere Application Server.

Example

Example 1

The following example deletes all the SPNEGO TAI properties for SPN2

```
wsadmin>$AdminTask deleteSpnegoTAIProperties {-spnId 2}
```

Example 2

The following example deletes all SPNEGO TAI properties

```
wsadmin>$AdminTask deleteSpnegoTAIProperties
com.ibm.ws.security.spnego.SPN1.filter=request-url!=noSPNEGO;request-url%=snoop
com.ibm.ws.security.spnego.SPN1.hostName=central01.austin.ibm.com
com.ibm.ws.security.spnego.SPN2.hostName=wssecpd.austin.ibm.com
wsadmin>
```

Displaying SPNEGO TAI properties using the wsadmin utility (deprecated):

You use the wsadmin utility to display the properties in the configuration of the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) trust association interceptor (TAI) for WebSphere Application Server.

About this task

Note:

> In WebSphere Application Server Version 6.1, a trust association interceptor (TAI) that uses the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) to securely negotiate and authenticate HTTP requests for secured resources was introduced. In WebSphere Application Server 7.0, this function is now deprecated. SPNEGO web authentication has taken its place to provide dynamic reload of the SPNEGO filters and to enable fallback to the application login method.

You use the wsadmin utility to configure the SPNEGO TAI for WebSphere Application Server:

Procedure

1. Start WebSphere Application Server.
2. Start the command-line utility by running the **wsadmin** command from the *app_server_root*/bin directory.
3. At the **wsadmin** prompt, enter the following command:
   ```
   $AdminTask showSpnegoTAIProperties
   ```

   You can use the following parameters with this command:

Option	Description
<spnId>	This is an optional parameter. It is the service principal name (SPN) identifier for the group of custom properties that are to be displayed with this command. If you do not specify this parameter, all SPNEGO TAI custom properties are displayed.

Results

SPNEGO TAI properties are displayed for this WebSphere Application Server.

Example

Example 1

The following example displays all SPNEGO TAI properties.

```
wsadmin>$AdminTask showSpnegoTAIProperties
com.ibm.ws.security.spnego.SPN1.filter=request-url!=noSPNEGO;request-url%=snoop
com.ibm.ws.security.spnego.SPN1.hostName=central01.austin.ibm.com
com.ibm.ws.security.spnego.SPN2.hostName=wssecpd.austin.ibm.com
wsadmin>
```

Example 2

The following example displays SPNEGO TAI properties for SPN1 and host, central01.austin.ibm.com.

```
wsadmin>$AdminTask showSpnegoTAIProperties -interactive
Show SPNEGO TAI configuration properties.

Display SPNEGO TAI configuration properties.

Service Principal Name identifier (spnId): 1

Show SPNEGO TAI configuration properties.

F (Finish)
C (Cancel)

Select [F, C]: [F]
WASX7278I: Generated command line: $AdminTask showSpnegoTAIProperties {-spnId 1}

com.ibm.ws.security.spnego.SPN1.filter=request-url!=noSPNEGO;request-url%=snoop
com.ibm.ws.security.spnego.SPN1.hostName=central01.austin.ibm.com
com.ibm.ws.security.spnego.SPN1.trimUserName=true
wsadmin>
```

SPNEGO TAI custom properties configuration (deprecated):

The Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) trust association interceptor (TAI) custom configuration properties control different operational aspects of the SPNEGO TAI. You can specify different property values for each application server.

Note:

In WebSphere Application Server Version 6.1, a trust association interceptor (TAI) that uses the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) to securely negotiate and authenticate HTTP requests for secured resources was introduced. In WebSphere Application Server 7.0, this function is now deprecated. SPNEGO web authentication has taken its place to provide dynamic reload of the SPNEGO filters and to enable fallback to the application login method.

Each of the properties defined in the following table is specified in the Custom Properties panel for the SPNEGO TAI using the administrative console facility. For convenience, you can optionally place these properties in a properties file. In this case, the SPNEGO TAI loads the configuration properties from the file instead of the Custom Properties panel definition. Refer to com.ibm.ws.security.spnego.propertyReloadFile property as defined in “SPNEGO TAI JVM configuration custom properties (deprecated)” on page 411.

To assign unique property names that identify each possible SPN, an SPN<id> is embedded in the property name and used to group the properties that are associated with each SPN. The SPN<id>s are numbered sequentially for each property group.

Table 30. SPNEGO TAI custom properties.

This table lists the SPNEGO TAI custom properties.

Property Name	Required	Default Value
“com.ibm.ws.security.spnego.SPN<id>.enableCredDelegate”	No	false
“com.ibm.ws.security.spnego.SPN<id>.filter”	No	See the description that follows.
“com.ibm.ws.security.spnego.SPN<id>.filterClass” on page 406	No	See the description that follows.
“com.ibm.ws.security.spnego.SPN<id>.hostName” on page 406	Yes	None
“com.ibm.ws.security.spnego.SPN<id>.NTLMTokenReceivedPage” on page 406	No	See the description that follows.
“com.ibm.ws.security.spnego.SPN<id>.spnegoNotSupportedPage” on page 406	No	See the description that follows.
“com.ibm.ws.security.spnego.SPN<id>.trimUserName” on page 406	No	true

Note: The following commands tasks can be used to operate on these SPNEGO TAI properties:

- “Adding SPNEGO TAI properties using the wsadmin utility (deprecated)” on page 396
- “Deleting SPNEGO TAI properties using the wsadmin utility (deprecated)” on page 401
- “Modifying SPNEGO TAI properties using the wsadmin utility (deprecated)” on page 399
- “Displaying SPNEGO TAI properties using the wsadmin utility (deprecated)” on page 402

com.ibm.ws.security.spnego.SPN<id>.enableCredDelegate:

This property is optional. It indicates whether or not the Kerberos delegated credentials are stored by the SPNEGO TAI. This property enables the capability for an application to retrieve the stored credentials and propagate them to other applications downstream for additional SPNEGO authentication.

This property requires use of the advanced Kerberos credential delegation feature and requires development of custom logic by the application developer. The developer must interact directly with the Kerberos Ticket Granting Service (TGS) to obtain a Ticket Granting Ticket (TGT) using the delegated Kerberos credentials on behalf of the end-user who originated the request. The developer must also construct the appropriate Kerberos SPNEGO token and include it in the HTTP request to continue the downstream SPNEGO authentication process, including handling additional SPNEGO challenge-response exchange, if necessary.

com.ibm.ws.security.spnego.SPN<id>.filter:

This property is optional. It defines the filtering criteria that is used by the specified class with the com.ibm.ws.security.spnego.SPN<id>.filterClass property. It defines arbitrary criteria that is meaningful to the implementation class used.

The com.ibm.ws.security.spnego.HTTPHeaderFilter default implementation class uses this property to define a list of selection rules that represent conditions that are matched against the HTTP request headers to determine whether or not the HTTP request is selected for SPNEGO authentication.

Each condition is specified with a key-value pair, separated from each other by a semicolon. The conditions are evaluated from left to right, as they display in the specified property. If all conditions are met, the HTTP request is selected for SPNEGO authentication.

The key and value in the key-value pair are separated by an operator that defines which condition is checked. The key identifies an HTTP request header to extract from the request and its value is compared with the value that is specified in the key-value pair according to the operator specification. If the header that is identified by the key is not present in the HTTP request, the condition is treated as not being met.

Any of the standard HTTP request headers can be used as the key in the key-value pairs. Refer to the HTTP specification for the list of valid headers. In addition, two keys are defined to extract information from the request, also useful as a selection criterion, which is not available through standard HTTP request headers. The remote-address key is used as a pseudo header to retrieve the remote TCP/IP address of the client application that sent the HTTP request. The request-URL key is used as a pseudo header to retrieve the URL that is used by the client application to make the request. The interceptor uses the result of the getRequestURL operation in the javax.servlet.http.HttpServletRequest interface to construct the web address. If a query string is present, the result of the getQueryString operation in the same interface is also used. In this case, the complete URL is constructed as follows:

```
String url = request.getRequestURL() + '?' + request.getQueryString();
```

The following operators and conditions are defined:

Table 31. Filter conditions and operations.

This table defines the conditions and operators used in filtering and gives examples.

Condition	Operator	Example
Match exactly	= = Arguments are compared as equal.	host=host.my.company.com
Match partially (includes)	%= Arguments are compared with a partial match being valid.	user-agent%=IE 6
Match partially (includes one of many)	^= Arguments are compared with a partial match being valid for one of many arguments specified.	request-url^=webApp1|webApp2|webApp3
Does not match	!= Arguments are compared as not equal.	request-url!=noSPNEGO
Greater than	> Arguments are compared lexogaphically as greater than.	remote-address>192.168.255.130
Less than	< Arguments are compared lexographically as less than.	remote-address<192.168.255.135

com.ibm.ws.security.spnego.SPN<id>.filterClass:

This property is optional. It specifies the name of the Java class that is used by the SPNEGO TAI to select which HTTP requests are subject to SPNEGO authentication.

If no class is specified, the default com.ibm.ws.security.spnego.HTTPHeaderFilter implementation class is used. The Java class that is specified must implement the `com.ibm.wsspi.security.spnego.SpnegoFilter` interface. A default implementation of this interface is provided. Specify the `com.ibm.ws.security.spnego.HTTPHeaderFilter` class to use the default implementation. This class uses the selection rules specified with the `com.ibm.ws.security.spnego.SPN<id>.filter` property.

com.ibm.ws.security.spnego.SPN<id>.hostName:

This property is required. It specifies the hostname in the SPN used by the SPNEGO TAI to establish a Kerberos secure context. It has no default value.

Note: The hostname is the long form of hostname. For example, `myHostName.austin.ibm.com`.
The Kerberos SPN is a string of the form `HTTP/hostname@realm`. The complete SPN is used with the Java Generic Security Service (JGSS) by the SPNEGO provider to obtain the security credential and security context that are used in the authentication process.

com.ibm.ws.security.spnego.SPN<id>.NTLMTokenReceivedPage:

This property is optional. It specifies the web address of a resource that contains the content that the SPNEGO TAI includes in the HTTP response that the (browser) client application displays when the SPNEGO token is received by the interceptor when the challenge-response handshake contains a NT LAN Manager (NTLM) token instead of the expected SPNEGO token.

It can specify a web (http://) or a file (file://) resource. If this property is not specified or the interceptor cannot find the specified resource, the following content is used:

```
<html><head><title>An NTLM Token was received.</title></head>
<body>Your browser configuration is correct, but you have not logged into a supported
Microsoft(R) Windows(R) Domain.
<p>Please login to the application using the normal login page.</html>
```

com.ibm.ws.security.spnego.SPN<id>.spnegoNotSupportedPage:

This property is optional. It specifies the web address of a resource that contains the content that the SPNEGO TAI includes in the HTTP response that the (browser) client application displays if it does not support SPNEGO authentication. It can specify a Web (http://) or a file (file://) resource.

If this property is not specified or the interceptor cannot find the specified resource, the following content is used:

```
<html><head><title>SPNEGO authentication is not supported</title></head>
<body>SPNEGO authentication is not supported on this client</body></html>;
```

com.ibm.ws.security.spnego.SPN<id>.trimUserName:

This property is optional. It specifies whether (`true`) or not (`false`) the SPNEGO TAI is to remove the suffix of the principal user name, starting from the "@" that precedes the Kerberos realm name.

If this property is set to `true`, the suffix of the principal user name is removed. If this property is set to `false`, the suffix of the principal name is retained. The default value used is `true`. For example,

When com.ibm.ws.security.spnego.SPN<id>.trimUserName = true

```
bobsmith@myKerberosRealm  becomes  bobsmith
```

When com.ibm.ws.security.spnego.SPN<id>.trimUserName = false

`bobsmith@myKerberosRealm remains bobsmith@myKerberosRealm`

SPNEGO TAI configuration requirements (deprecated):

The configuration that is used by the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) trust association interceptor (TAI) on each selected application server is governed by various system requirements.

Note:

> In WebSphere Application Server Version 6.1, a trust association interceptor (TAI) that uses the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) to securely negotiate and authenticate HTTP requests for secured resources was introduced. In WebSphere Application Server 7.0, the SPNEGO TAI was deprecated. SPNEGO web authentication has taken its place to provide dynamic reload of the SPNEGO filters and to enable fallback to the application login method.

The following list of configuration requirements highlights those attributes, properties, qualities, restrictions, exclusions, inclusions, and dependencies that you need to be aware of when planning a WebSphere Application Server configuration that incorporates the use of the SPNEGO TAI.

Table 32. SPNEGO TAI requirements.

This table lists the SPNEGO TAI configuration requirements.

Function item	Description
SPNEGO TAI	The SPNEGO TAI is a server side solution in WebSphere Application Server. Client-side applications are responsible for generating the SPNEGO token for use by the SPNEGO TAI.
Microsoft Windows	Microsoft Windows Servers with Active Directory domain and its associated Kerberos key distribution center (KDC) is required. For information on the supported Microsoft Windows Servers, see the System Requirements for WebSphere Application Server Version 8.5 on Windows.
Client application (browser or .NET client)	A browser (client application) or .NET client that supports the SPNEGO authentication mechanism, as defined in IETF RFC 2478 is required.
Simple and Protected GSS-API Negotiation Mechanism (SPNEGO)	SPNEGO authentication, as defined in IETF RFC 2478 is used.
Internet browsers	• Use Microsoft Internet Explorer version 5.5 or higher • Use Mozilla Firefox version 1.0
Kerberos Level	Kerberos version 5 is required.
WebSphere Application Server	Version 7.0 is required.
Java SDK level	Java 6.0 SDK is required.
Encryption Types	RC4-HMAC encryption is only supported when using a Windows 2003 Server as Kerberos key distribution center (KDC).
J2EE client	Client application (browser or .NET client) A browser (client application) or .NET client that supports the SPNEGO authentication mechanism, as defined in IETF RFC 2478 is required.

Configuring the client browser to use SPNEGO TAI (deprecated)

You can configure your browser to utilize the Simple and Protected GSS-API Negotiation (SPNEGO) mechanism. Authentication of your browser requests are processed by the SPNEGO trust association interceptor (TAI) in the WebSphere Application Server.

Before you begin

You need to know how to display and set options in the Microsoft Internet Explorer browser or any other browser (such as Firefox). You must have a browser installed that supports SPNEGO authentication.

Note:

In WebSphere Application Server Version 6.1, a trust association interceptor (TAI) that uses the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) to securely negotiate and authenticate HTTP requests for secured resources was introduced. In WebSphere Application Server 7.0, this function is now deprecated. SPNEGO web authentication has taken its place to provide dynamic reload of the SPNEGO filters and to enable fallback to the application login method.

About this task

Complete the following steps to ensure that your Microsoft Internet Explorer browser is enabled to perform SPNEGO authentication.

Procedure

1. At the desktop, log in to the windows active directory domain.
2. Activate Internet Explorer.
3. In the Internet Explorer window, click **Tools > Internet Options > Security** tab.
4. Select the **Local intranet** icon and click **Sites**.
5. In the Local intranet window, ensure that the "check box" to include all local (intranet) not listed in other zones is selected, then click **Advanced**.
6. In the **Local intranet** window, fill in the Add this web site to the zone field with the web address of the host name so that the single sign-on (SSO) can be enabled for the list of websites shown in the websites field. Your site information technology staff provides this information. Click **OK** to complete this step and close the Local intranet window.
7. On the **Internet Options** window, click the **Advanced** tab and scroll to **Security settings**. Ensure that the **Enable Integrated Windows Authentication (requires restart)** box is selected.
8. Click **OK**. Restart your Microsoft Internet Explorer to activate this configuration.

Results

Complete the following steps to ensure that your Firefox browser is enabled to perform SPNEGO authentication.

1. At the desktop, log in to the windows active directory domain.
2. Activate Firefox.
3. At the address field, type **about:config**.
4. In the Filter, type **network.n**
5. Double click on **network.negotiate-auth.trusted-uris**. This preference lists the sites that are permitted to engage in SPNEGO Authentication with the browser. Enter a comma-delimited list of trusted domains or URLs.

   Note: You must set the value for **network.negotiate-auth.trusted-uris**.
6. If the deployed SPNEGO solution is using the advanced Kerberos feature of Credential Delegation double click on **network.negotiate-auth.delegation-uris**. This preference lists the sites for which the browser may delegate user authorization to the server. Enter a comma-delimited list of trusted domains or URLs.
7. Click **OK**. The configuration appears as updated.

8. Restart your Firefox browser to activate this configuration.

Your Internet browser is properly configured for SPNEGO authentication. You can use applications that are deployed in WebSphere Application Server that use secured resources without being repeatedly requested for an ID and password.

Configuring JVM custom properties, filtering HTTP requests, and enabling SPNEGO TAI in WebSphere Application Server (deprecated)

Performing this task helps you, as web administrator, to ensure that WebSphere Application Server is configured to enable the operation of the Simple and Protected GSS-API Negotiation mechanism (SPNEGO) trust association interceptor (TAI) with the required Java virtual machine (JVM) property and with the appropriate filtering of HTTP requests.

Before you begin

You need to know how to use the WebSphere Application Server administrative console to manage the security configuration and have the proper authority to modify the security configuration of the application server.

Note:

> In WebSphere Application Server Version 6.1, a trust association interceptor (TAI) that uses the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) to securely negotiate and authenticate HTTP requests for secured resources was introduced. In WebSphere Application Server 7.0, this function is now deprecated. SPNEGO web authentication has taken its place to provide dynamic reload of the SPNEGO filters and to enable fallback to the application login method.

About this task

Verify the configuration of your SPNEGO TAI. The deployment of the SPNEGO TAI can vary from a single WebSphere Application Server system on which a single application is running to a large multinode WebSphere Application Server, Network Deployment (ND) cell, with dozens of application servers, hosting many applications. Every SPNEGO TAI is installed at the cell level. You must be aware of your particular SPNEGO TAI configuration.

The default behavior of the SPNEGO TAI is to not intercept HTTP requests. This default behavior ensures that the SPNEGO TAI can be installed into an existing cell, configured for a single application server and not change any other application servers in the cell. Other WebSphere Application Servers can run exactly as before within a given configuration.

Decide whether or not to use the sample SPN<id>.filterClass and determine the exact filter properties to use.

Note: The default behavior of the SPNEGO TAI is to use the `com.ibm.ws.security.spnego.SPN<id>.filterClass` and intercept all requests.

If the default behavior is not appropriate, you can use a customer provided class, or extend or modify the sample class as required. The system programmer interface, `com.ibm.ws.security.spnego.SpnegoFilter` allows you to implement a custom filter to determine whether or not to intercept a particular HTTP request. With the default implementation, you can set filter rules for coarse as well as fine-grained criteria in selecting which HTTP requests to intercept.

Note: For an alternative to the steps below for enabling the SPNEGO TAI, you can use scripting to perform the operation. See “Enabling the SPNEGO TAI as JVM custom property using scripting (deprecated)” on page 410 for the details.

Complete the following steps to enable the operation of the SPNEGO TAI with your selected filtering and with the JVM required property.

Procedure

1. Log on to WebSphere Application Server administrative console.
2. Click **Servers > Application servers**.
3. Select the appropriate server. Under Server Infrastructure, expand **Java and process management > Process Definition**.
4. Click **Java virtual machine**. Under Additional Properties, click **Custom Properties**. Create a new custom property, if required, by clicking **New**, then code `com.ibm.ws.security.spnego.isEnabled` in the name field and `true` in the value field.
5. Click **Apply > OK** to save the configuration
6. Identify when the SPNEGO TAI intercepts a given request. A set of filter properties is provided, but you must determine what is appropriate and modify the `com.ibm.ws.security.spnego.SPN<id>.filterClass` accordingly.

Results

The application server is configured and ready to provide a single sign-on environment for end users who have successfully authenticated in a Microsoft Active Directory domain. You must restart each application server that is configured for SPNEGO web authentication. Then your SPNEGO TAI is set to filter HTTP request when it is operating.

Enabling the SPNEGO TAI as JVM custom property using scripting (deprecated):

You use the wsadmin utility to enable the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) trust association interceptor (TAI) for WebSphere Application Server.

Before you begin

Before starting this task, the wsadmin tool must be running. See the information about starting the wsadmin scripting client using wsadmin scripting.

Note:

> In WebSphere Application Server Version 6.1, a trust association interceptor (TAI) that uses the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) to securely negotiate and authenticate HTTP requests for secured resources was introduced. In WebSphere Application Server 7.0, this function is now deprecated. SPNEGO web authentication has taken its place to provide dynamic reload of the SPNEGO filters and to enable fallback to the application login method.

About this task

Perform the following steps to enable the SPNEGO TAI:

Procedure

1. Identify the server and assign it to the `server1` variable:
   - Using Jacl:
     ```
     set server1 [$AdminConfig getid /Cell:mycell/Node:mynode/Server:server1/]
     ```
   - Using Jython:
     ```
     server1 = AdminConfig.getid("/Cell:mycell/Node:mynode/Server:server1/")
     print server1
     ```

Example output:

```
server1(cells/mycell/nodes/mynode|servers/seerver1|server.xml#Server_1)
```

2. Identify the Java virtual machine (JVM) belonging to this server and assign it to the `jvm` variable:
   - Using Jacl:

     ```
     set jvm [$AdminConfig list JavaVirtualMachine $server1]
     ```

   - Using Jython:

     ```
     jvm = AdminConfig.list('JavaVirtualMachine',server1)
     ```

   Example output:

   ```
   (cells/mycell/nodes/mynode/servers/server1:server.xml#JavaVirtualMachine_1)
   (cells/mycell/nodes/mynode/servers/server1:server.xml#JavaVirtualMachine_2)
   ```

3. Identify the controller JVM of the server:
   - Using Jacl:

     ```
     set cjvm [lindex $jvm 0]
     ```

   - Using Jython:

     ```
     # get line separator
     import java
     lineSeparator = java.lang.System.getProperty('line.separator')
     arrayJVMs = jvm.split(lineSeparator)
     cjvm = arrayJVMs[0]
     ```

4. Modify the generic JVM arguments to enable SPNEGO TAI:
   - Using Jacl:

     ```
     set attr_name           [list name com.ibm.ws.security.spnego.isEnabled]
     set attr_value          [list value true]
     set attr_required       [list required false]
     set attr_description    [list description "Enabled SPNEGO TAI"]

     set attrs [list $attr_name $attr_value $attr_required $attr_description]

     $AdminConfig create Property $cjvm $attrs
     ```

   - Using Jython:

     ```
     attr_name  = ['name', "com.ibm.ws.security.spnego.isEnabled"]
     attr_value = ['value', "true"]
     attr_required = ['required', "false"]
     attr_description = ['description', "Enabled SPNEGO TAI"]
     attr_list = [attr_name, attr_value, attr_required, attr_description]
     property=['systemProperties',[attr_list]]
     AdminConfig.modify(cjvm, [property])
     ```

5. Save the configuration changes.

SPNEGO TAI JVM configuration custom properties (deprecated):

Java virtual machine (JVM) custom properties control the operation of the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) trust association interceptor (TAI).

Note:

> In WebSphere Application Server Version 6.1, a trust association interceptor (TAI) that uses the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) to securely negotiate and authenticate HTTP requests for secured resources was introduced. In WebSphere Application Server 7.0, this function is now deprecated. SPNEGO web authentication has taken its place to provide dynamic reload of the SPNEGO filters and to enable fallback to the application login method.

The following JVM custom properties control operation of the SPNEGO TAI. Different custom property values can be specified for each application server.

Table 33. JVM configuration custom properties.

This table lists the SPNEGO JVM configuration custom properties.

Custom Property Name	Required	Value Type	Default Value	Recommended Value
com.ibm.ws.security.spnego.isEnabled	No	Boolean	False	True

com.ibm.ws.security.spnego.propertyReloadFile	No	String	None	For Windows
				`C:\temp\TAI.props`
				
				For UNIX
				`/tmp/TestTAI.Properties`

com.ibm.ws.security.spnego.propertyReloadTimeout	No	Integer	None	120
com.ibm.ws.security.spnego.useHttpFilterClass2	No	Boolean	False	True

com.ibm.ws.security.spnego.isEnabled
Use this custom property to enable or disable operation of the SPNEGO TAI in a given application server. When set to `false`, the SPNEGO TAI is disabled and not used by the web authentication module for authenticating any web requests. When set to `true`, the SPNEGO TAI is enabled and used by the web authentication module for authenticating any web requests.

com.ibm.ws.security.spnego.propertyReloadFile
Use this custom property to identify the file that contains configuration properties for the SPNEGO TAI, when it is not convenient to stop and restart the application server. The properties contained in this file can be reloaded to configure the SPNEGO TAI.

Important: The properties that are defined in the specified file override any properties defined using the administrative console.

A sample of this reload file follows:

```
#########################################################
# Template properties files for SPNEGO TAI
#
# Where possible defaults have been provided.
#
#########################################################

#---------------------------------------------------------
# Hostname
#---------------------------------------------------------
#com.ibm.ws.spnego.SPN1.HostName=wsecurity.austin.ibm.com

#---------------------------------------------------------
# (Optional) SpnegoNotSupportedPage
#---------------------------------------------------------
#com.ibm.ws.spnego.SPN1.SpnegoNotSupportedPage=

#---------------------------------------------------------
# (Optional) NTLMTokenReceivedPage
#---------------------------------------------------------
#com.ibm.ws.spnego.SPN1.NTLMTokenReceivedPage=

#---------------------------------------------------------
# (Optional) FilterClass
#---------------------------------------------------------
#com.ibm.ws.spnego.SPN1.FilterClass=com.ibm.ws.spnego.HTTPHeaderFilter

#---------------------------------------------------------
# (Optional) Filter
#---------------------------------------------------------
#com.ibm.ws.spnego.SPN1.Filter=
```

Important: If com.ibm.ws.security.spnego.propertyReloadFile custom property is set, but the com.ibm.ws.security.spnego.propertyReloadTimeout custom property is not, then the SPNEGO TAI is not initialized.

com.ibm.ws.security.spnego.propertyReloadTimeout
Use this custom property to specify a time interval in seconds that elapses after which the SPNEGO TAI reloads the configuration properties. Also, the SPNEGO TAI reloads the configuration properties if the file that is identified by the com.ibm.ws.security.spnego.propertyReloadFile custom property changed since the last time the configuration custom properties were retrieved. This time interval in seconds must be specified as a positive integer.

com.ibm.ws.security.spnego.useHttpFilterClass2
Use this custom property to specify that the HttpHeaderFilter classes should be used. The HttpHeaderFilter classes enable:

- The `!=` operator to be used for SPNEGO TAI filters.
- A space to exist in a SPNEGO TAI filter.

When this property is set to `true` the following filter specification works properly.

```
user-agent!=IBM Web Services Explorer;request-url!=noSPNEGO
```

If this property is set to `false`, or is not specified, the preceding filter does not work properly.

Important:

- If the com.ibm.ws.security.spnego.propertyReloadFile custom property and the com.ibm.ws.security.spnego.propertyReloadTimeout custom property are not set, then the SPNEGO TAI properties are only loaded once from the SPNEGO TAI custom properties defined in the WebSphere Application Server configuration data. This one time loading occurs when the JVM is initialized.
- If com.ibm.ws.security.spnego.propertyReloadTimeout custom property is set, but the com.ibm.ws.security.spnego.propertyReloadFile custom property is not, then the SPNEGO TAI is not initialized. “Configuring JVM custom properties, filtering HTTP requests, and enabling SPNEGO TAI in WebSphere Application Server (deprecated)” on page 409 or how to configure the JVM custom properties for SPNEGO TAI.

Remember: You can also use the wsadmin command for the AdminConfig scripting object to interactively set the com.ibm.ws.security.spnego.isEnabled custom property. See “Enabling the SPNEGO TAI as JVM custom property using scripting (deprecated)” on page 410 for more information.

The following custom properties are not used directly by the SPNEGO TAI; however, they affect the operation of the core security runtime and can also be used for problem determination.

Table 34. JVM configuration custom properties.

This table describes the JVM configuration custom properties

Custom Property Name	Required	Value Type	Default Value	Recommended Value
com.ibm.security.jgss.debug	No	String	None	"off" or "all"
com.ibm.security.krb5.Krb5Debug	No	String	None	"off" or "all"
java.security.properties	No	String	None	
javax.security.auth.useSubjectCredsOnly	Yes	Boolean	True	False

com.ibm.security.jgss.debug
This custom property is optional. It can be used to collect diagnostic trace information for problem determination in the Java Generic Security Service (JGSS) application programmer interface (API) implementation. The value can be set to `all` or `off` to enable or disable tracing, respectively. See Java Generic Security Service User's Guide for specific JGSS API information.

com.ibm.security.krb5.Krb5Debug
This custom property is optional. It can be used to collect additional diagnostic trace information for problem determination in the JGSS implementation. The value can be set to `all` or `off` to enable or disable tracing, respectively.

java.security.properties
This property is optional. It can be used when different application servers in a cell have different security requirements and it is not convenient to modify the global `java.security` file for the entire cell. In such situations, the java.security.properties custom property is used to specify the location of the `java.security` file used by the JVM for each application server.

javax.security.auth.useSubjectCredsOnly
JGSS includes an optional Java Authentication and Authorization Service (JAAS) login facility that saves `Principal` credentials and secret keys in the `Subject` of the application's JAAS login context. JGSS retrieves credentials and secret keys from the `Subject` by default. This feature can be disabled by setting the Java property javax.security.auth.useSubjectCredsOnly to `false`.

Attention: The SPNEGO TAI does not use the optional JAAS login module. The javax.security.auth.useSubjectCredsOnly property must be set to `false`.

Mapping Kerberos client principal name to WebSphere user registry ID for SPNEGO TAI (deprecated)

You can use a system programming interface to customize the behavior of the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) trust association interceptor (TAI) by implementing arbitrary mappings of the end-user's identity, which is retrieved from Microsoft Active Directory to the identity that is used in the WebSphere Application Server security registry.

Before you begin

You need to perform some administrative tasks in the WebSphere Application Server environment to use SPNEGO TAI and to ensure that the requester's identity matches the identity in the WebSphere Application Server user registry.

Note:

In WebSphere Application Server Version 6.1, a trust association interceptor (TAI) that uses the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) to securely negotiate and authenticate HTTP requests for secured resources was introduced. In WebSphere Application Server 7.0, this function is now deprecated. SPNEGO web authentication has taken its place to provide dynamic reload of the SPNEGO filters and to enable fallback to the application login method.

Note: Make sure the following tasks have been performed successfully:

1. Configuring the web browser to use SPNEGO. See “Configuring the client browser to use SPNEGO TAI (deprecated)” on page 407
2. Configuring Java virtual machine (JVM) properties, custom SPNEGO TAI properties, and enabling the SPNEGO TAI. See “Configuring JVM custom properties, filtering HTTP requests, and enabling SPNEGO TAI in WebSphere Application Server (deprecated)” on page 409

About this task

In the simplest deployment of the SPNEGO TAI, it is assumed that the requester's identity in the WebSphere Application Server user registry is identical to the identity retrieved. This is the case when Microsoft Windows Active Directory server is the lightweight directory access protocol (LDAP) server used in WebSphere Application Server. This is default behavior of the SPNEGO TAI.

You do not need to use this simple deployment of the SPNEGO TAI. WebSphere Application Server can use a different registry, such as a local OS, LDAP, or custom registry instead of the Microsoft Active Directory. If WebSphere Application Server uses a different registry than the Microsoft Active Directory, then a mapping from the Microsoft Windows user Id to a WebSphere Application Server user Id is necessary.

Procedure

Use the JAAS custom login module to perform any custom mapping of a client Kerberos principal name from the Microsoft Active Directory to the WebSphere user registry identity. The JAAS custom login module is a plug-in mechanism that is defined for authenticating incoming and outgoing requests in WebSphere Application Server and is inserted before the `ltpaLoginModule`. The JAAS custom login module retrieves a client Kerberos principal name in the `javax.security.auth.Subject` using `subject.getPrincipals(KerberosPrincipal.class)` method, maps the client Kerberos principal name to the WebSphere user registry identity, and inserts the mapping identity in the hash table property `com.ibm.wsspi.security.cred.userId`. The `ltpaLoginModule` then uses the mapped identity to create a `WSCredential`.

Note: The custom login module can also supply the full set of security properties in the `javax.security.auth.Subject` in the `com.ibm.wsspi.security.tai.TAIResult` to fully assert the mapped identity. When the identity is fully asserted, the `wsMapDefaultInboundLoginModule` maps those security properties to a `WSCredential`.

A sample of the custom login module follows:

```
package com.ibm.ws.security.server.lm;

import java.util.Map;
import java.lang.reflect.Array;
import javax.security.auth.Subject;
import javax.security.auth.callback.*;
import javax.security.auth.login.LoginException;
import javax.security.auth.spi.LoginModule;
import javax.security.auth.kerberos.*;
import com.ibm.websphere.security.auth.WSLoginFailedException;
import com.ibm.wsspi.security.token.AttributeNameConstants;

/**
 *
 * @author IBM Corporation
 * @version 1.0
 * @since 1.0
 *
 */

public class sampleSpnegoMappingLoginModule implements LoginModule {
    /*
     *
     * Constant that represents the name of this mapping module.  Whenever this sample
     * code is used to create a class with a different name, this value should be changed.
     *
     */
    private final static String MAPPING_MODULE_NAME = "com.ibm.websphere.security.sampleSpnegoMappingLoginModule";

    private String mapUid = null;
    /**
     * Construct an uninitialized WSLoginModuleImpl object.
     */
    public sampleSpnegoMappingLoginModule() {
        debugOut("sampleSpnegoMappingLoginModule() entry");
        debugOut("sampleSpnegoMappingLoginModule() exit");
    }

    /**
```

```
 * Initialize this login module.
 *
 *
 * This is called by the  LoginContext after this login module is
 * instantiated. The relevant information is passed from the LoginContext
 * to this login module. If the login module does not understands any of the data
 * stored in the sharedState and options parameters,
 * they can be ignored.
 *
 *
 * @param subject The subject to be authenticated.
 * @param callbackHandler
 *                A  CallbackHandler for communicating with the end user to gather
 *                login information (e.g., username and password).
 * @param sharedState
 *                The state shared with other configured login modules.
 * @param options The options specified in the login configuration for this particular login module.
 */
public void initialize(Subject subject, CallbackHandler callbackHandler,
                       Map sharedState, Map options) {
    debugOut("initialize(subject = \"" + subject.toString() +
             "\", callbackHandler = \"" + callbackHandler.toString() +
             "\", sharedState = \"" + sharedState.toString() +
             "\", options = \"" + options.toString() + "\")");

    this.subject = subject;
    this.callbackHandler = callbackHandler;
    this.sharedState = sharedState;
    this.options = options;

    debug = "true".equalsIgnoreCase((String)this.options.get("debug"));

    debugOut("initialize() exit");
}

/**
 *
 * Method to authenticate a Subject (phase 1).
 *
 *
 *
 * This method authenticates a Subject. It uses CallbackHandler to gather
 * the Subject information, like username and password for example, and verify these
 * information. The result of the authentication is saved in the private state within
 * this login module.
 *
 *
 * @return  true if the authentication succeeded, or false
 *         if this login module should be ignored.
 * @exception LoginException
 *                   If the authentication fails.
 */
public boolean login() throws LoginException
{
    debugOut("sampleSpnegoMappingLoginModule.login() entry");

    boolean succeeded = false;
    java.util.Set krb5Principals= subject.getPrincipals(KerberosPrincipal.class);
    java.util.Iterator krb5PrincIter = krb5Principals.iterator();

    while (krb5PrincIter.hasNext()) {
        Object princObj = krb5PrincIter.next();
        debugOut("Kerberos principal name: "+ princObj.toString());

        if (princObj != null && princObj.toString().equals("utle@WSSEC.AUSTIN.IBM.COM")){
            mapUid = "user1";
            debugOut("mapUid: "+mapUid);
```

```
                            java.util.Hashtable customProperties = (java.util.Hashtable)
                            sharedState.get(AttributeNameConstants.WSCREDENTIAL_PROPERTIES_KEY);
                            if (customProperties == null) {
                                customProperties = new java.util.Hashtable();
                            }
                            succeeded = true;
                            customProperties.put(AttributeNameConstants.WSCREDENTIAL_USERID, mapUid);

                            Map<String,java.util.Hashtable)>
                            mySharedState=(Map<String,java.util.Hashtable>)sharedState;
                            mySharedState.put((AttributeNameConstants.WSCREDENTIAL_PROPERTIES_KEY.customProperties)

                            debugOut("Add a mapping user ID to Hashtable, mapping ID = "+mapUid);

                               }
                            debugOut("login() custom properties = " + customProperties);
        }
    }

    succeeded = true;
    debugOut("sampleSpnegoMappingLoginModule.login() exit");

    return succeeded;
}

/**
 *
 * Method to commit the authentication result (phase 2).
 *
 *
 *
 * This method is called if the LoginContext's overall authentication
 * succeeded (the revelant REQUIRED, REQUISITE, SUFFICIENT and OPTIONAL login module
 * succeeded).
 *
 *
 * @return true if the commit succeeded, or false
 *         if this login module should be ignored.
 * @exception LoginException
 *                   If the commit fails.
 */
public boolean commit() throws LoginException
{
    debugOut("commit()");

    debugOut("commit()");

    return true;
}

/**
 * Method to abort the authentication process (phase 2).
 *
 *
 * This method is called if the LoginContext's overall authentication
 * failed (the revelant REQUIRED, REQUISITE, SUFFICIENT and OPTIONAL login module
 * did not succeed).
 *
 *
 *
 * If this login module's authentication attempt succeeded, then this method cleans
 * up the previous state saved in phase 1.
 *
 *
 * @return true if the abort succeeded, or false
 *         if this login module should be ignored.
```

```
     * @exception LoginException
     *                  If the abort fails.
     */
    public boolean abort() throws LoginException {
        debugOut("abort() entry");
        debugOut("abort() exit");
        return true;
    }

    /**
     * Method which logs out a Subject.
     *
     * @return true if the logout succeeded, or false
     *         if this login module should be ignored.
     * @exception LoginException
     *                  If the logout fails.
     */
    public boolean logout() throws LoginException
    {
        debugOut("logout() entry");
        debugOut("logout() exit");

        return true;
    }

    private void cleanup()
    {
        debugOut("cleanup() entry");
        debugOut("cleanup() exit");
    }

    /*
     *
     * Private method to print trace information.  This implementation uses System.out
     * to print trace information to standard output, but a custom tracing system can
     * be implemented here as well.
     *
     */
    private void debugOut(Object o)
    {
        System.out.println("Debug: " + MAPPING_MODULE_NAME);
        if (o != null) {
            if (o.getClass().isArray()) {
                int length = Array.getLength(o);
                for (int i = 0; i < length; i++) {
                    System.out.println("\t" + Array.get(o, i));
                }
            } else {
                System.out.println("\t" + o);
            }
        }
    }
    private Subject subject;
    private CallbackHandler callbackHandler;
    private Map sharedState;
    private Map options;

    protected boolean debug = false;
}
```

Results

Using the custom login module, Microsoft Active Directory identities are mapped to the WebSphere Application Server's security registry and the behavior of the SPNEGO TAI is customized.

Single sign-on capability with SPNEGO TAI - checklist (deprecated)

WebSphere Application Server provides a trust association interceptor (TAI) that uses the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) to securely negotiate and authenticate HTTP requests for secured resources in WebSphere Application Server. To deploy and use the SPNEGO TAI you need to examine your installation and decide on how best to configure the SPNEGO TAI.

Note:

In WebSphere Application Server Version 6.1, a trust association interceptor (TAI) that uses the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) to securely negotiate and authenticate HTTP requests for secured resources was introduced. In WebSphere Application Server 7.0, this function is now deprecated. SPNEGO web authentication has taken its place to provide dynamic reload of the SPNEGO filters and to enable fallback to the application login method.

Lightweight Third Party Authentication (LTPA) is the default authentication mechanism for WebSphere Application Server. However, you may need to configure LTPA prior to configuring the SPNEGO TAI. LTPA is the required authentication mechanism for all trust association interceptors. You can configure LTPA by clicking **Security > Global security > Authentication mechanisms and expiration**.

Note: Enabling web security single sign-on (SSO) is optional when you configure the SPNEGO TAI. For more information, see “Implementing single sign-on to minimize web user authentications” on page 373.

Answer the following questions to establish how the SPNEGO TAI is deployed.

1. What is your criteria for intercepting HTTP requests?

   You must decide if the SPNEGO TAI deployment will use the HTTPHeaderFilter class as the default. If you do use this class, then you must specify the exact filter properties for this class. The default behavior of the SPNEGO TAI is to use the com.ibm.ws.spnego.HTTPHeaderFilter class to intercept all requests.

   If you do not use the sample com.ibm.ws.spnego.HTTPHeaderFilter class, then you must define a new class that implements the com.ibm.wsspi.security.spnego.SpnegoTAIFilter interface.

   You can decide to further control what HTTP requests are intercepted using the Service Provider Programming Interface (SPI), “Filtering HTTP requests for SPNEGO TAI (deprecated)” on page 420

   See “SPNEGO TAI custom properties configuration (deprecated)” on page 403 for descriptions of

   - com.ibm.ws.security.spnego.SPN<id>.filterClass
   - com.ibm.ws.security.spnego.SPN<id>.filter

2. Is user Id mapping to be used? If not, why not?

   WebSphere Application Server enables you to define or develop a custom login module to map user IDs. See “Mapping Kerberos client principal name to WebSphere user registry ID for SPNEGO TAI (deprecated)” on page 414 for more detail about performing this mapping.

   You must decide, before deploying the TAI, whether or not to use this custom login module to perform the SPNEGO TAI identity mapping

3. What type of encryption is to be used to process the SPNEGO tokens?

   Microsoft Windows Active Directory supports two different Kerberos encryption types: RC4-HMAC and DES-CBC-MD5. The IBM Java Generic Security Service (JGSS) library (and SPNEGO library) support both of these encryption types.

   Restriction: RC4-HMAC encryption is only supported with a Windows 2003 Server key distribution center (KDC).

4. How will you handle credential delegation?

Kerberos supports the delegation of credentials. A server that receives Kerberos credentials from a client can impersonate that client to other servers by using delegated credentials. Since SPNEGO TAI tokens are a wrapping of a Kerberos credential, a server that receives Kerberos credentials within an SPNEGO token can use those Kerberos credentials to impersonate the original user. That server can interact using SPNEGO over HTTP as a SPNEGO client to other SPNEGO servers by composing an appropriate HTTP Authorization header.

5. Will the SPNEGO TAI be deployed in a single or multiple domain name service (DNS) domain environment?

   Web browsers running on Windows are sensitive to DNS domains. They only send a SPNEGO token when the target host name identifies a host name defined in the DNS domain of the client machine. You can use HTTP redirection to support this configuration with the creation of a pseudo Kerberos service principal name (SPN) in each DNS domain. All SPNs that WebSphere Application Server supports must have their secret keys available in Kerberos `keytab` files. To enable single sign-on across multiple DNS domains, a separate Kerberos keytab file is generated for each SPN per domain. These individual Kerberos keytab files must be merged before they can be used by WebSphere Application Server.

6. How frequently will application servers reload the SPNEGO TAI properties?

   The SPNEGO TAI has an optional property reload feature that allows the reloading of the TAI properties without restarting the Java virtual machine (JVM). This reload feature is controlled by the system properties `com.ibm.ws.security.spnego.propertyReloadFile` and `com.ibm.ws.security.spnego.propertyReloadTimeout`. These properties taken together enable the SPNEGO TAI internal properties to be reloaded from a file on the file system after a certain time period. If the com.ibm.ws.security.spnego.propertyReloadTimeout attribute is set to a valid integer value, and the com.ibm.ws.security.spnego.propertyReloadFile attribute points to a file on the file system, then each JVM reloads the SPNEGO TAI properties from the file after the timeout period expires. Also, the SPNEGO TAI properties are reloaded only if the date on the file has changed. If these reload properties are not set, then the SPNEGO TAI properties are only loaded once, at JVM initialization, from the SPNEGO TAI custom properties that are defined in WebSphere Application Server configuration data. See “SPNEGO TAI JVM configuration custom properties (deprecated)” on page 411 for more information about these reload properties.

The Windows Active Directory (Web) administrator, the WebSphere Application Server administrator, and the application team review and answer these questions to determine the best deployment and configuration settings for the SPNEGO TAI.

Filtering HTTP requests for SPNEGO TAI (deprecated)

You can use a system programming interface to customize the behavior of the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) trust association interceptor (TAI) by specifying whether or not a particular HTTP request should be intercepted.

Before you begin

Before you begin, you need to understand the deployment of the SPNEGO TAI in your installation.

Note:

In WebSphere Application Server Version 6.1, a trust association interceptor (TAI) that uses the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) to securely negotiate and authenticate HTTP requests for secured resources was introduced. In WebSphere Application Server 7.0, this function is now deprecated. SPNEGO web authentication has taken its place to provide dynamic reload of the SPNEGO filters and to enable fallback to the application login method.

About this task

Verify the configuration of your SPNEGO TAI. The deployment of the SPNEGO TAI can vary from a single WebSphere Application Server system on which a single application is running to a large multinode WebSphere Application Server, Network Deployment (ND) cell, with dozens of application servers, hosting many applications. Every SPNEGO TAI is installed at the cell level. You must be aware of your particular SPNEGO TAI configuration.

The default behavior of the SPNEGO TAI is to not intercept HTTP requests. This default behavior ensures that the SPNEGO TAI can be installed into an existing cell, configured for a single application server and not change any other application servers in the cell. Other WebSphere Application Servers can run exactly as before within a given configuration.

Then decide whether or not to use the sample SPN<id>.filter class and determine the exact filter properties to use.

Note: The default behavior of the SPNEGO TAI is to use the com.ibm.ws.security.spnego.SPN<id>.filter class and intercept all requests.

If the default behavior is not appropriate, you can use a customer provided class, or extend or modify the sample class as required. The system programmer interface, com.ibm.ws.security.spnego.SpnegoFilter allows you to implement a custom filter to determine whether or not to intercept a particular HTTP request. With the default implementation, you can set filter rules for coarse as well as fine-grained criteria in selecting which HTTP requests to intercept.

Procedure

1. Set the `com.ibm.ws.security.spnego.isEnabled` Java virtual machine (JVM) custom property to `true` to enable the SPNEGO TAI on any JVM.
2. Identify when the SPNEGO TAI intercepts a given request. A set of filter properties is provided, but you must determine what is appropriate and modify the `com.ibm.ws.security.spnego.SPN<id>.filter class` accordingly.

Results

Your SPNEGO TAI is set to filter HTTP requests when it is operating.

Configuring single sign-on capability with Tivoli Access Manager or WebSEAL

Use the following information to enable single sign-on to WebSphere Application Server using either WebSEAL or the plug-in for web servers.

About this task

Either Tivoli Access Manager WebSEAL or Tivoli Access Manager plug-in for web servers can be used as reverse proxy servers to provide access management and single sign-on (SSO) capability to WebSphere Application Server resources. With such an architecture, either WebSEAL or the plug-in authenticates users and forwards the collected credentials to WebSphere Application Server in the form of an IV Header. Two types of single sign-on are available, the TAI interface and the TAI++ interface, so named as both use WebSphere Application Server trust association interceptors (TAI). With the TAI, the end-user name is extracted from the HTTP header and forwarded to embedded Tivoli Access Manager where the end-user name is used to construct the client credential information and authorize the user. With the TAI++, all of the user credential information is available in the HTTP header and not just the user name. The TAI++ is the more efficient of the two solutions because a Lightweight Directory Access Protocol (LDAP) call is not required. TAI functionality is retained for backwards compatibility.

Complete the following tasks to enable single sign-on to WebSphere Application Server using either WebSEAL or the plug-in for web servers. These tasks assume that embedded Tivoli Access Manager is configured for use.

Procedure

1. Create a trusted user account for Tivoli Access Manager in the shared Lightweight Directory Access Protocol (LDAP) user registry. For more information, see “Creating a trusted user account in Tivoli Access Manager” on page 428.
2. Configure either WebSEAL or the Tivoli Access Manager plug-in for Web servers to work with WebSphere Application Server. For more information, see either of the following articles:
   - “Configuring WebSEAL for use with WebSphere Application Server” on page 429
   - “Configuring Tivoli Access Manager plug-in for web servers for use with WebSphere Application Server” on page 429
3. Configure single sign-on using either the TAI or TAI++ interface. For more information, see either of the following articles:
   - “Configuring single sign-on using trust association” on page 430
   - “Configuring single sign-on using trust association interceptor ++” on page 431

Single sign-on settings

Use this page to set the configuration values for single sign-on (SSO).

To view this administrative console page, complete the following steps:

1. Click **Security** > **Global security**.
2. Under **Authentication**, click **Web and SIP security** > **Single sign-on (SSO)**.

The Set security cookies as HTTPOnly to resist cross-site scripting attacks check box has been added to the Single sign-on settings page for this release. The HttpOnly attribute is a browser attribute created to prevent client side applications (such as Java scripts) from accessing cookies to prevent some cross-site scripting vulnerabilities. The attribute specifies that LTPA and WASReqURL cookies include the HTTPOnly field.

Enabled:

Specifies that the single sign-on function is enabled.

Web applications that use J2EE FormLogin style login pages, such as the administrative console, require single sign-on (SSO) enablement. Only disable SSO for certain advanced configurations where LTPA SSO-type cookies are not required.

Information	Value
Data type:	Boolean
Default:	Enabled
Range:	Enabled or Disabled

Requires SSL:

Specifies that the single sign-on function is enabled only when requests are made over HTTPS Secure Sockets Layer (SSL) connections. When this property is enabled, security is automatically enabled.

Information	Value
Data type:	Boolean
Default:	Disable
Range:	Enable or Disable

Domain name:

Specifies the domain name (.ibm.com, for example) for all single sign-on hosts.

The application server uses all the information after the first period, from left to right, for the domain names. If this field is not defined, the web browser defaults the domain name to the host name where the web application is running. Also, single sign-on is then restricted to the application server host name and does not work with other application server host names in the domain.

You can specify multiple domains separated by a semicolon (;), a space ( ), a comma (,), or a pipe (|). Each domain is compared with the host name of the HTTP request until the first match is located. For example, if you specify ibm.com;austin.ibm.com and a match is found in the ibm.com domain first, the application server does not match the austin.ibm.com domain. However, if a match is not found in either ibm.com or austin.ibm.com, then the application server does not set a domain for the LtpaToken cookie.

gotcha: The session manager uses a secure random generator to generate session ID. The session ID is written to the cookie when the cookie is created in the setCookie method. The session manager does not set the LtpaToken to cookies.

If you specify the `UseDomainFromURL` value, the application server sets the SSO domain name value to the domain of the host that is used in the web address. For example, if an HTTP request comes from `server1.raleigh.ibm.com`, the application server sets the SSO domain name value to `raleigh.ibm.com`.

Tip: The UseDomainFromURL value is case insensitive. You can type `usedomainfromurl` to use this value.

Information	Value
Data type:	String

Interoperability mode:

Specifies that an interoperable cookie is sent to the browser to support back-level servers.

In WebSphere Application Server, Version 6 and later, a new cookie format is needed by the security attribute propagation functionality. When the interoperability mode flag is enabled, the server can send a maximum of two single sign-on (SSO) cookies back to the browser. In some cases, the server just sends the interoperable SSO cookie.

Web inbound security attribute propagation:

When web inbound security attribute propagation is enabled, security attributes are propagated to front-end application servers. When this option is disabled, the single sign-on (SSO) token is used to log in and recreate the Subject from the user registry.

With this information, the receiving server can contact the originating server using an MBean call to get the original serialized security attributes.

Set security cookies as HTTPOnly to resist cross-site scripting attacks:

The HttpOnly attribute is a browser attribute created to prevent client side applications (such as Java scripts) from accessing cookies to prevent some cross-site scripting vulnerabilities. The attribute specifies that LTPA and WASReqURL cookies include the HTTPOnly field.

For session cookies, see the session settings for servers, applications, and web modules.

Information	Value
Data type:	boolean

Information	Value
Default:	enabled
Range:	enabled or disabled

com.tivoli.pd.jcfg.PDJrteCfg utility for Tivoli Access Manager single sign-on

The com.tivoli.pd.jcfg.PDJrteCfg utility configures the Java Runtime Environment component for Tivoli Access Manager. This utility enables Java applications to use the Tivoli Access Manager policy and authorization servers.

Purpose

Syntax

```
java com.tivoli.pd.jcfg.PDJrteCfg -action {config | unconfig} -host policy_server_host -was -java_home jre_path
```

Parameters

-action {config|unconfig}
Specifies the action to be performed. Actions include:

config Use to configure the Access Manager Java Runtime Environment component.

unconfig
Use to reconfigure the Access Manager Java Runtime Environment component.

-cfgfiles_path
Specifies where the generated configuration files will be placed.

Note: This parameter is required.

-host *policy_server_host*
Specifies the policy server host name.

Valid values for *policy_server_host* include any valid IP host name.

Examples include:

```
host = libra
host = libra.dallas.ibm.com
```

-was
Notifies Tivoli Access Manager Runtime for Java that the WebSphere Application Server version is being configured so it is not necessary to perform certain steps such as copying the Java security jar files and PD.jar file since they were already placed in the appropriate directory by the WebSphere Application Server installer.

-java_home *jre_path*
Specifies the fully qualified path to the Java runtime (such as the directory ending in jre). If this parameter is not specified, the home directory for the jre in the PATH statement is used. If the home directory for the jre is not in the PATH statement, this utility can create an incorrect parameter in the output files.

Comments

This command copies Tivoli Access Manager Java libraries to a library extensions directory that exists for a Java runtime that has already been installed on the system.

You can install more than one Java Runtime Environment (JRE) on a given machine. The `pdjrtecfg` command can be used to configure the Tivoli Access Manager Java Runtime Environment component independently for each of the JRE configurations.

```
${JAVA_HOME}/bin/java
-Dfile.encoding=ISO8859-1 \
-Xnoargsconversion \
-Dpd.home=${WAS_HOME}/java/jre/PolicyDirector \
-cp /zWebSphere/w8z.na1/AppServer/tivoli/tam/PD.jar \
 com.tivoli.pd.jcfg.PDJrteCfg \
-action config \
          -cfgfiles_path ${WAS_HOME}/java/jre \
          -host gary.us.ibm.com \
          -was
          -java_home ${JAVA_HOME}
```

where:

-Dws.output.encoding
Is used to enable z/OS to display all of its messages and errors in a readable format.

-Dpd.home
Indicates where Tivoli Access Manager Runtime for Java has been installed. For WebSphere Application Server, this is `java.home/PolicyDirector`

com.tivoli.pd.jcfg.SvrSslCfg utility for Tivoli Access Manager single sign-on

The utility is used to configure and remove the configuration information associated with WebSphere Application Server and the Tivoli Access Manager server.

Purpose

Syntax

```
java com.tivoli.pd.jcfg.SvrSslCfg
-action {config | unconfig} -admin_id admin_user_ID
-admin_pwd admin_password -appsvr_id application_server_name
-appsvr_pwd application_server_password -mode{local|remote}
-host host_name_of_application_server
-policysvr policy_server_name:port:rank [,...]
-authzsvr authorization_server_name:port:rank [,...]
-cfg_file fully_qualified_name_of_configuration_file
-domain Tivoli_Acccess_Manager_domain
-key_file fully_qualified_name_of_keystore_file
-cfg_action {create|replace}
```

Parameters

-action {config | unconfig}
Specifies the configuration action that is performed by the script. The following options apply:

-action config
Configuring a server creates user and server information in the user registry and creates local configuration and key store files on the application server. Use the `-action unconfig` option to reverse this operation.

If this action is specified, the following options are required: `-admin_id`, `-admin_pwd`, `-appsvr_id`, `-port`, `-mode`, `-policysvr`, `-authzsvr`, and `-key_file`.

-action unconfig
Reconfigures an application server to complete the following actions:

- Remove the user and server information from the user registry
- Delete the local key store file
- Remove information for this application from the configuration file without deleting the file

The reconfiguration operation fails only if the caller is unauthorized or the policy server cannot be contacted.

This action can succeed when a configuration file does not exist. When the configuration file does not exist, it is created and used as a temporary file to hold configuration information during the operation, and then the file is deleted completely.

If this action is specified, the following options are required: `-admin_id`, `-admin_pwd`, `-appsvr_id`, and `-policysvr`.

-admin_id *admin_user_ID*
Specifies the Tivoli Access Manager administrator name. If this option is not specified, `sec_master` is the default.

A valid administrative ID is an alphanumeric, case-sensitive string. String values are expected to be characters that are part of the local code set. You cannot use a space in the administrative ID.

For example, for U.S. English the valid characters are the letters `a-Z`, the numbers `0-9`, a period (`.`), an underscore (_), a plus sign (+), a hyphen (-), an at sign (`@`), an ampersand (`&`), and an asterisk (`*`). The minimum and maximum lengths of the administrative ID, if there are limits, are imposed by the underlying registry.

-admin_password *admin_password*

Specifies the password of the Tivoli Access Manager administrator user that is associated with the **-admin_id** parameter. The password restrictions depend upon the password policy for your Tivoli Access Manager configuration.

-appsvr_id *application_server_name*
Specifies the name of the application server. The name is combined with the host name to create unique names for Tivoli Access Manager objects created for your application. The following names are reserved for Tivoli Access Manager applications: `ivacld`, `secmgrd`, `ivnet`, and `ivweb`.

-appsvr_pwd *application_server_password*
Specifies the password of the application server. This option is required. A password is created by the system and the configuration file is updated with the password created by the system.

If this option is not specified, the server password will be read from standard input.

-authzsvr *authorization_server_name*
Specifies the name of the Tivoli Access Manager authorization server with which the application server communicates. The server is specified by fully qualified host name, the SSL port number, and the rank. The default SSL port number is 7136. For example: `myauth.mycompany.com:7136:1`. You can specify multiple servers if the entries are separated by a comma (,).

-cfg_action {create | replace}
Specifies the action to take when creating the configuration and key files. Valid values are **create** or **replace**. Use the **create** option to initially create the configuration and keystore files. Use the **replace** option if these files already exist. If you use the **create** option and the configuration or keystore files already exist, an exception is created.

Options are as follows:

create Specifies to create the configuration and key store files during server configuration. Configuration fails if either of these files already exists.

replace
Specifies to replace the configuration and key store files during server configuration. Configuration deletes any existing files and replaces them with new ones.

-cfg_file *fully_qualified_name_of_configuration_file*
Specifies the configuration file path and name.

A file name should be an absolute file name (fully qualified file name) to be valid.

-domain *Tivoli_Access_Manager_domain*
Specifies the Tivoli Access Manager domain name to which the administrator is authenticated. This domain must exist and an the administrator ID and password must be valid for this domain. The application server is specified in this domain.

If not specified, the local domain that was specified during Tivoli Access Manager runtime configuration will be used. The local domain value will be retrieved from the configuration file.

A valid domain name is an alphanumeric, case-sensitive string. String values are expected to be characters that are part of the local code set. You cannot use a space in the domain name.

For example, for U.S. English the valid characters for domain names are the letters a-Z, the numbers 0-9, a period (.), an underscore (_), a plus sign (+), a hyphen (-), an at sign (@), an ampersand (&), and an asterisk (*). The minimum and maximum lengths of the domain name, if there are limits, are imposed by the underlying registry.

-host *host_name_of_application_server*
Specifies the TCP host name used by the Tivoli Access Manager policy server to contact this server. This name is saved in the configuration file using the azn-app-host key.

The default is the local host name returned by the operating system. Valid values for host_name include any valid IP host name.

Examples:

```
host = libra
host = libra.dallas.ibm.com
```

-key_file *fully_qualified_name_of_keystore_file*
Specifies the directory that is to contain the key files for the server. A valid directory name is determined by the operating system. Use a fully qualified file name that contains the application server certificate and key file.

Make sure that server user (for example, `ivmgr`) or all users have permission to access the `.kdb` file and the folder that contains the `.kdb` file.

This option is required.

-mode *server_mode*
Specifies the mode in which the application operates. This value must be either `local` or `remote`.

-policysvr *policy_server_name*
Specifies the name of the policy server.

Comments

After the successful configuration of a Tivoli Access Manager Java application server, `SvrSslCfg` creates a user account and server entries representing the Java application server in the Tivoli Access Manager user registry. In addition, `SvrSslCfg` creates a configuration file and a Java key store file, which securely stores a client certificate, locally on the application server. This client certificate permits callers to make authenticated use of Tivoli Access Manager services. Conversely, reconfiguration removes the user and server entries from the user registry and cleans up the local configuration and keystore files.

The contents of an existing configuration file can be modified by using the `SvrSslCfg` utility. The configuration file and the key store file must already exist when calling `SvrSslCfg` with all options other than `-action config` or `-action unconfig`.

The following options are parsed and processed into the configuration file, but are otherwise ignored in this version of Tivoli Access Manager:

The host name is used to build a unique name (identity) for the application. The `pdadmin` user list command displays the application identity name in the following format:

server_name/host_name

Note that the `pdadmin` server list command displays the server name in a slightly different format:

server_name-host_name

```
CLASSPATH=${WAS_HOME}/tivoli/tam/PD.jar:${WAS_CLASSPATH}
java \
-cp ${CLASSPATH} \
-Dpd.cfg.home= ${WAS_HOME}/java/jre \
-Dfile.encoding=ISO8859-1 \
-Xnoargsconversion \
 com.tivoli.pd.jcfg.SvrSslCfg \
-action config \
-admin_id sec_master \
-admin_pwd $TAM_PASSWORD \
-appsvr_id $APPSVR_ID \
-policysvr ${TAM_HOST}:7135:1 \
-port 7135 \
-authzsvr ${TAM_HOST}:7136:1 \
-mode remote \
-cfg_file ${CFG_FILE} \
-key_file ${KEY_FILE} \
-cfg_action create
```

Creating a trusted user account in Tivoli Access Manager

Tivoli Access Manager trust association interceptors require the creation of a trusted user account in the shared LDAP user registry.

About this task

This account includes the ID and password that WebSEAL uses to identify itself to WebSphere Application Server. To prevent potential vulnerabilities, do not use the sec_master ID as the trusted user account and ensure that the password you use is unique and generated randomly. Use the trusted user account for the TAI or TAI++ only.

Procedure

1. Use either the Tivoli Access Manager pdadmin command-line utility or Web Portal Manager to create the trusted user. For example, from the **pdadmin** command line.
2. Reference the code listed below as an example for creating a trusted user account.
3. Reference the following additional resources for more information:
   a. “Configuring WebSEAL for use with WebSphere Application Server” on page 429
   b. “Configuring Tivoli Access Manager plug-in for web servers for use with WebSphere Application Server” on page 429

Example

```
pdadmin> user create webseal_userid webseal_userid_DN firstname
         surname password

pdadmin> user modify webseal_userid account-valid yes
```

Configuring WebSEAL for use with WebSphere Application Server

Use this topic to set the SSO password in WebSEAL for single sign-on to WebSphere Application Server.

About this task

A junction must be created between WebSEAL and WebSphere Application Server. This junction carries the iv-credentials (for TAI++) or iv-user (for TAI) and the HTTP basic authentication headers with the request. You can configure WebSEAL to pass the end user identity in other ways, the iv-credentials header is the only one supported by the TAI++ and the iv-user is the only one supported by TAI.

Communications over the junction should use Secure Sockets Layer (SSL) for increased security. Setting up SSL across this junction requires that you configure the HTTP Server used by WebSphere Application Server, and WebSphere Application Server itself, to accept inbound SSL traffic and route it correctly to WebSphere Application Server. This activity requires importing the necessary signing certificates into the WebSEAL certificate keystore, and possibly also the HTTP Server certificate keystore.

Create the junction between WebSEAL and WebSphere Application Server using the **-c iv_creds** option for TAI++ and **-c iv_user** for TAI. Enter either of the following commands as one line using the variables that are appropriate for your environment:

TAI++

```
server task webseald-server create -t ssl -b supply -c iv_creds
 -h host_name -p websphere_app_port_number junction_name
```

TAI

```
server task webseald-server create -t ssl -b supply -c iv_user
 -h host_name -p websphere_app_port_number junction_name
```

Notes:

1. If warning messages are displayed about the incorrect setup of certificates and key databases, delete the junction, correct problems with the key databases, and recreate the junction.
2. The junction can be created as `-t tcp` or `-t ssl`, depending on your requirements.

For single sign-on (SSO) to WebSphere Application Server the SS) password must be set in WebSEAL. To set the password, complete the following steps:

Procedure

1. Edit the WebSEAL configuration file *webseal_install_directory*/etc/webseald-default.conf Set the following parameter: `basicauth-dummy-passwd=`*webseal_userid_passwd*

   where *webseal_userid_passwd* is the SSO password for the trusted user account set in “Creating a trusted user account in Tivoli Access Manager” on page 428.
2. Restart WebSEAL.

What to do next

For more details and options about how to configure junctions between WebSEAL and WebSphere Application Server, including other options for specifying the WebSEAL server identity, refer to the *Tivoli Access Manager WebSEAL Administration Guide* as well as to the documentation for the HTTP Server you are using with your WebSphere Application Server. Tivoli Access Manager documentation is available at http://publib.boulder.ibm.com/tividd/td/tdprodlist.html.

Configuring Tivoli Access Manager plug-in for web servers for use with WebSphere Application Server

Tivoli Access Manager plug-in for web servers can be used as a security gateway for your protected WebSphere Application Server resources.

About this task

With such an arrangement the plug-in authorizes all user requests before passing the credentials of the authorized user to WebSphere Application Server in the form of an iv-creds header. Trust between the plug-in and WebSphere Application Server is established through use of basic authentication headers containing the single sign-on (SSO) user password.

Procedure

1. The Tivoli Access Manager plug-in for web servers configuration shows IV headers configured for post-authorization processing, and basic authentication that is configured as the authentication mechanism and for post-authorization processing, as shown in the example below.
2. After a request is authorized, the basic authentication header is removed from the request (`strip-hdr=always`) and a new one is added (`add-hdr=supply`).
3. Included in this new header is the password that is set when the SSO user is created in “Creating a trusted user account in Tivoli Access Manager” on page 428.
4. Specify this password in the **supply-password** parameter and it is passed in the newly created header. This basic authentication header enables trust between WebSphere Application Server and the plug-in.
5. An iv-creds header is also added (`generate=iv-creds`), which contains the credential information of the user passed onto WebSphere Application Server. Session cookies are used to maintain session state.

Example

```
[common-modules]
authentication = BA
session = session-cookie
post-authzn = BA
post-authzn = iv-headers

[iv-headers]
accept = all
generate = iv-creds

[BA]
strip-hdr = always
add-hdr = supply
supply-password = sso_user_password
```

What to do next

“Configuring single sign-on using trust association” or “Configuring single sign-on using trust association interceptor ++” on page 431

Configuring single sign-on using trust association

This task is performed to enable single sign-on using trust association. Trust association is used to connect reversed proxy servers to the application server.

Before you begin

Note: Use of TAIs for Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) authentication is deprecated in this release. The SPNEGO web authentication panels provide a much easier and less error-prone way to configure SPNEGO.

To establish the trust association for the single sign-on, perform the following steps:

Procedure

1. From the administrative console for WebSphere Application Server, click **Security > Global security**.
2. From Authentication mechanisms, click **Web and SIP security > Trust association**.
3. Select the **Enable trust association** option.
4. Under Additional properties, click the **Interceptors** link.

5. Click **com.ibm.ws.security.web.TAMTrustAssociationInterceptorPlus** to use a WebSEAL interceptor, or **com.ibm.ws.security.spnego.TrustAssociationInterceptorImpl** to use a SPNEGO interceptor.
6. Under Custom properties, select a custom property to edit or click **New** to create a new one. Enter the property name and value pairs.
7. Click **OK**.
8. Save the configuration and log out.
9. Restart WebSphere Application Server.

Configuring single sign-on using trust association interceptor ++

Perform this task to enable single sign-on using trust association interceptor ++. The steps involve setting up trust association and creating the interceptor properties.

Before you begin

Lightweight Third Party Authentication (LTPA) is the default authentication mechanism for WebSphere Application Server. However, you may need to configure LTPA prior to configuring the TAMTrustAssociationInterceptorPlus. LTPA is the required authentication mechanism for all trust association interceptors. You can configure LTPA by clicking **Security** > **Global security** > **Authentication mechanisms and expiration**.

Note: Enabling web security single sign-on (SSO) is optional when you configure the TAMTrustAssociationInterceptorPlus. For more information, see “Implementing single sign-on to minimize web user authentications” on page 373.

Although you can use Simple WebSphere Authentication Mechanism (SWAM) by selecting the **Use SWAM-no authenticated communication between servers** option on the Authentication mechanisms and expiration panel, single sign-on (SSO) requires LTPA as the configured authentication mechanism.

To establish the trust association for the single sign-on, perform the following steps:

Procedure

1. From the administrative console for WebSphere Application Server, click **Security Global security** .
2. Under Web security, click **Trust association**.
3. Click **Enable Trust Association**.
4. Click **Interceptors**.
5. Click **com.ibm.ws.security.web.TAMTrustAssociationInterceptorPlus** to use a WebSEAL interceptor. This interceptor is one of two WebSEAL interceptors that are supplied for your use. You choose to use this interceptor by supplying properties as described in the next step.

   Attention: WebSphere Application Server attempts to initialize both of these interceptors even if you only supplied properties for the **com.ibm.ws.security.web.TAMTrustAssociationInterceptorPlus** interceptor. As a result, messages AWXRB0008E and SECJ0384E can appear during initialization to indicate that the interceptor you did not choose has failed to initialize. This is normal processing and does not affect the initialization of the interceptor you did select. To inhibit the display of messages AWXRB0008E and SECJ0384E, you can delete the interceptor you do not want to use prior to beginning the initialization. You can add that interceptor back later if your environment changes.
6. Click **Custom Properties**.
7. Click **New** to enter the property name and value pairs. Ensure that the following parameters are set:

Table 35. Custom properties.

This table describes the TAI custom properties.

Option	Description
com.ibm.websphere.security.webseal.checkViaHeader	You can configure TAI so that the via header can be ignored when validating trust for a request. Set this property to *false* if none of the hosts in the via header need to be trusted. When set to *false* you do not need to set the trusted host names and host ports properties. The only mandatory property to check when via header is *false* is com.ibm.websphere.security.webseal.loginId. The default value of the check via header property is *false*. When using Tivoli Access Manager plug-in for web servers, set this property to *false*. **Note:** The via header is part of the standard HTTP header that records the server names the request that passed through.
com.ibm.websphere.security.webseal.loginId	The WebSEAL trusted user as created in “Creating a trusted user account in Tivoli Access Manager” on page 428 The format of the username is the short name representation. This property is mandatory. If it is not set in WebSphere Application Server, the TAI initialization fails.
com.ibm.websphere.security.webseal.id	A comma-separated list of headers that exists in the request. If all of the configured headers do not exist in the request, trust cannot be established. The default value for the ID property is *iv-creds*. Any other values set in WebSphere Application Server are added to the list along with iv-creds, separated by commas.
com.ibm.websphere.security.webseal.hostnames	Do not set this property if using Tivoli Access Manager Plug-in for Web Servers. The property specifies the host names (case sensitive) that are trusted and expected in the request header. Requests arriving from un-listed hosts might not be trusted. If the checkViaHeader property is not set or is set to false then the trusted host names property has no influence. If the checkViaHeader property is set to `true`, and the trusted host names property is not set, TAI initialization fails.
com.ibm.websphere.security.webseal.ports	Do not set this property if using Tivoli Access Manager plug-in for web servers. This property is a comma-separated list of trusted host ports. Requests that arrive from unlisted ports might not be trusted. If the checkViaHeader property is not set, or is set to `false` this property has no influence. If the checkViaHeader property is set to `true`, and the trusted host ports property is not set in WebSphere Application Server, the TAI initialization fails.

com.ibm.websphere.security.webseal.viaDepth	A positive integer that specifies the number of source hosts in the via header to check for trust. By default, every host in the via header is checked, and if any host is not trusted, trust cannot be established. The via depth property is used when only some of the hosts in the via header have to be trusted. The setting indicates the number of hosts that are required to be trusted.
	
	As an example, consider the following header:
	`Via: HTTP/1.1 webseal1:7002, 1.1 webseal2:7001`
	
	If the viaDepth property is not set, is set to 2 or is set to 0, and a request with the previous via header is received then both `webseal1:7002` and `webseal2:7001` need to be trusted. The following configuration applies:
	`com.ibm.websphere.security.webseal.hostnames = webseal1,webseal2`
	`com.ibm.websphere.security.webseal.ports = 7002,7001`
	
	If the via depth property is set to 1, and the previous request is received, then only the last host in the via header needs to be trusted. The following configuration applies:
	`com.ibm.websphere.security.webseal.hostnames = webseal2`
	`com.ibm.websphere.security.webseal.ports = 7001`
	
	The viaDepth property is set to `0` by default, which means all of the hosts in the via header are checked for trust.

com.ibm.websphere.security.webseal.ssoPwdExpiry	After trust is established for a request, the single sign-on user password is cached, eliminating the need to have the TAI re-authenticate the single sign-on user with Tivoli Access Manager for every request. You can modify the cache timeout period by setting the single sign-on password expiry property to the required time in seconds. If the password expiry property is set to `0`, the cached password never expires. The default value for the password expiry property is `600`.
com.ibm.websphere.security.webseal.ignoreProxy	This property can be used to tell the TAI to ignore proxies as trusted hosts. If set to `true` the comments field of the hosts entry in the via header is checked to determine if a host is a proxy. Remember that not all proxies insert comments in the via header indicating that they are proxies. The default value of the ignoreProxy property is `false`. If the checkViaHeader property is set to `false` then the ignoreProxy property has no influence in establishing trust.
com.ibm.websphere.security.webseal.configURL	For the TAI to establish trust for a request, it requires that the SvrSslCfg run for the Java Virtual Machine on the Application Server and result in the creation of a properties file. If this properties file is not at the default URL, which is `file://java.home/PdPerm.properties`, the correct URL of the properties file must be set in the configuration URL property. If this property is not set, and the SvrSslCfg-generated properties file is not in the default location, the TAI initialization fails. The default value for the config URL property is `file://$WAS_HOME/java/jre/PdPerm.properties`.

8. Click **OK**.
9. Save the configuration and log out.

10. Restart WebSphere Application Server.

Configuring global sign-on principal mapping

You can create a new application login that uses the Tivoli Access Manager GSO database to store the login credentials.

Procedure

1. Click **Security > Global security**.
2. Under Authentication, click **Java Authentication and Authorization Service > Application logins**.
3. Click **New** to create a new Java Authentication and Authorization Service (JAAS) login configuration.
4. Enter the alias name of the new application login. Click **Apply**.
5. Under Additional properties, click **JAAS login modules** to define the JAAS Login Modules.
6. Click **New** and enter the following information:

   Module class name: com.tivoli.pdwas.gso.AMPrincipalMapper
   Use Login Module Proxy: enable
   Authentication strategy: REQUIRED
7. Click **Apply**
8. Under Additional Properties section, click **Custom Properties** to define login module-specific values that are passed directly to the underlying login modules.
9. Click **New**.

   The Tivoli Access Manager principal mapping module uses the `authDataAlias` configuration string to retrieve the correct user name and password from the security configuration.

   The authDataAlias attribute that is passed to the module is configured for the J2C connection factory. Because the authDataAlias attribute is an arbitrary string that is entered at configuration time, the following scenarios are possible:

   - The authDataAlias attribute contains both the global sign-on (GSO) resource name and the user name. The format of this string is `"Resource/User"`.
   - The authDataAlias attribute contains the GSO Resource name only. The user name is determined by using the Subject of the current session.

   The scenario to use is determined by a JAAS configuration option, as shown here:

   Name: com.tivoli.pd.as.gso.AliasContainsUserName

   Value: True, if the alias contains the user name; false, if the user name must be retrieved from the security context

   When entering `authDataAlias` attributes through the WebSphere Application Server administrative console, the node name is automatically pre-pended to the alias. The JAAS configuration entry determines whether this node name is removed or included as part of the resource name, as shown here:

   Name: com.tivoli.pd.as.gso.AliasContainsNodeName

   Value: True, if the alias contains the node name

   Note: If the `PdPerm.properties` configuration file is not located in the `JAVA_HOME/PdPerm.properties` default location, then you also need to add the following property:

   Name: com.tivoli.pd.as.gso.AMCfgURL

   Value: file:///*path to PdPerm.properties*

   Enter each new parameter using the following scenario information as a guide, then click **Apply**.

   Scenario 1
   Auth Data Alias - BackendEIS/eisUser

Resource - BackEndEIS
User - eisUser
Principal Mapping Parameters

Table 36. Principal Mapping Parameters.

This table lists the principal mapping parameters.

Name	Value
delegate	com.tivoli.pdwas.gso.AMPrincipalMapper
com.tivoli.pd.as.gso.AliasContainsUserName	true
com.tivoli.pd.as.gso.AliasContainsNodeName	false
com.tivoli.pd.as.gso.AMLoggingURL	file:///*jlog_props_path*
debug	false

Scenario 2
Auth Data Alias - BackendEIS
Resource - BackEndEIS
User - Currently authenticated WebSphere Application Server user
Principal Mapping Parameters

Table 37. Principal Mapping Parameters.

This table lists the principal mapping parameters.

Name	Value
delegate	com.tivoli.pdwas.gso.AMPrincipalMapper
com.tivoli.pd.as.gso.AliasContainsUserName	false
com.tivoli.pd.as.gso.AliasContainsNodeName	false
com.tivoli.pd.as.gso.AMLoggingURL	file:///*jlog_props_path*
debug	false

Scenario 3
Auth Data Alias - nodename/BackendEIS/eisUser
Resource - BackEndEIS
User - eisUser
Principal Mapping Parameters

Table 38. Principal Mapping Parameters.

This table lists the principal mapping parameters.

Name	Value
delegate	com.tivoli.pdwas.gso.AMPrincipalMapper
com.tivoli.pd.as.gso.AliasContainsUserName	true
com.tivoli.pd.as.gso.AliasContainsNodeName	true
com.tivoli.pd.as.gso.AMLoggingURL	file:///*jlog_props_path*
debug	false

Scenario 4
Auth Data Alias - nodename/BackendEIS/eisUser
Resource - nodename/BackEndEIS (notice that node name is not removed)
User - eisUser
Principal Mapping Parameters

Table 39. Principal Mapping Parameters.

This table lists the principal mapping parameters.

Name	Value
delegate	com.tivoli.pdwas.gso.AMPrincipalMapper
com.tivoli.pd.as.gso.AliasContainsUserName	true
com.tivoli.pd.as.gso.AliasContainsNodeName	false
com.tivoli.pd.as.gso.AMLoggingURL	file:///*jlog_props_path*
debug	false

Scenario 5
Auth Data Alias - BackendEIS/eisUser
Resource - BackEndEIS
User - eisUser
Principal Mapping Parameters

Table 40. Principal Mapping Parameters.

This table lists the principal mapping parameters.

Name	Value
delegate	com.tivoli.pdwas.gso.AMPrincipalMapper
com.tivoli.pd.as.gso.AliasContainsUserName	false
com.tivoli.pd.as.gso.AliasContainsNodeName	true
com.tivoli.pd.as.gso.AMLoggingURL	file:///*jlog_props_path*
debug	false

Scenario 6
Auth Data Alias - nodename/BackendEIS/eisUser
Resource - nodename/BackendEIS/eisUser
(notice that the resource is the same as Auth Data Alias).
User - Currently authenticated WebSphere Application Server user
Principal Mapping Parameters

Table 41. Principal Mapping Parameters.

This table lists the principal mapping parameters.

Name	Value
delegate	com.tivoli.pdwas.gso.AMPrincipalMapper
com.tivoli.pd.as.gso.AliasContainsUserName	false
com.tivoli.pd.as.gso.AliasContainsNodeName	false
com.tivoli.pd.as.gso.AMLoggingURL	file:///*jlog_props_path*
debug	false

10. Create the Java 2 Connector (J2C) authentication aliases. The user name and password that are assigned to these alias entries are irrelevant because Tivoli Access Manager is responsible for providing user names and passwords. However, the user name and password that are assigned to the J2C authentication aliases need to exist so that they can be selected for the J2C connection factory in the administrative console.

    To create the J2C authentication aliases, from the WebSphere Application Server administrative console, click **Security Global security**. Under Authentication, click **Java Authentication and Authorization Service J2C authentication data**, and then click **New** for each new entry. Refer to the previous table for scenario inputs.

    The connection factories for each resource adapter that need to use the GSO database must be configured to use the Tivoli Access Manager Principal mapping module:

a. From the WebSphere Application Server administrative console, click **Applications Enterprise Applications application_nameResourcer references**. Note that J2C connection factories must be already configured for the selected application. To configure a new J2C connection factory, see the Configuring Java EE Connector connection factories in the administrative console article.
b. Under Additional properties, click **Resource Adapter**.

   The resource adapter can be stand-alone and does not need to be packaged with the application. The resource adapter is configured from **Resources Resource Adapters** for stand-alone scenarios.
c. Under Additional properties, click **J2C Connection Factories**.
d. Click **New** and enter the connection factory properties.
e. When finished, click **Apply Save**.

Attention:
Custom mapping configuration for the connection factory is deprecated in WebSphere Application Server Version 6. To configure the GSO credential mapping, use the Map Resource References to Resources panel on the administrative console. For more information, see the J2EE connector security article.

Configuring administrative authentication

An authentication mechanism defines rules about security information, such as whether a credential is forwardable to another Java process, and the format of how security information is stored in both credentials and tokens. The Rivest Shamir Adleman (RSA) token authentication mechanism simplifies the security environment for flexible management topology, that is, the topology where you can locally or remotely submit and manage administrative jobs through a job manager that manages applications, perform product maintenance, modify configurations, and control the application server runtime. You use the administrative console to configure administrative authentication, which involves the configuring of the Rivest Shamir Adleman (RSA) token authentication mechanism.

Before you begin

The following keystore, truststore, and rootstore descriptions give you an idea of where certificates are stored and how trust is configured between processes.

The **NodeRSATokenKeyStore** contains the Rivest Shamir Adleman (RSA) token personal certificate used for this process. Not only is the public/private key from this certificate used to create RSA tokens, but the public key is used by other processes to create tokens. The RSA personal certificate is signed by an RSA root certificate.

The **NodeRSATokenTrustStore** contains all RSA signer certificates from other processes that are trusted to send RSA tokens to this process. The signers in this trust store are placed there automatically during the registration process. However, this task allows an administrator to configure trust between to processes not normally involved in the same administrative domain. There may be requirements where two base servers are communicating administratively. When using the RSA token authentication mechanism, the base servers need to share RSA signers if administrative communications is operating in both directions.

The NodeRSATokenRootStore contains the root personal certificate that is used to create new RSA personal certificates. Do not use the root certificate to create RSA tokens because this usage compromises the long-lived keys. Only use the root certificate to sign other certificates.

No manual steps are required with these keystores, and this allows uncommon trust establishment among processes not in the same administrative domain. You can also replace the RSA personal certificate with a personal certificate obtained from a certificate authority (CA) if desired. In this case, make sure the CA root certificate is placed in all RSA trust stores in the same administrative domain.

Procedure

1. Click **Security** > **SSL certificate and key management**.
2. Under **Related Items**, click **Key stores and certificates**.
3. Under **Keystore usages**, select **RSA token keystores**.
4. Select the RSA token key store you want to administer.
5. Modify the description if required.
6. Modify the path if required.
7. Select **read only**, **initialize at setup**, or both if required.
8. Enter the correct password to make these modifications
9. Click **Apply** and **Save**.

Results

You configured administrative authentication.

What to do next

In cases where the process is back-level or a target RSA certificate cannot be obtained, the fallback mechanism is Lightweight Third-Party Authentication (LTPA) which is supported in all previous releases for administrative communications. The fallback occurs automatically. If the LTPA keys are not shared and a fallback occurs, LTPA will fail as well. However, this situation is typically an error case in the RSA mechanism and should occur infrequently.

Java Authentication and Authorization Service

The standard Java 2 security application programming interface (API) helps enforce access control based on the location of the code source or the author or packager of the code that signed the jar file. The current principal of the running thread is not considered in the Java 2 security authorization. Instances where authorization is based on the principal, as opposed to the code base, and the user exist. The Java Authentication and Authorization Service is a standard Java API that supports the Java 2 security authorization to extend the code base on the principal as well as the code base and users.

The Java Authentication and Authorization Service (JAAS) Version 1.0 extends the Java 2 security architecture of the Java 2 platform with additional support to authenticate and enforce access control with principals and users. JAAS implements a Java version of the standard Pluggable Authentication Module (PAM) framework, and extends the access control architecture of the Java 2 platform in a compatible fashion to support user-based authorization or principal-based authorization. WebSphere Application Server fully supports the JAAS architecture. JAAS extends the access control architecture to support role-based authorization for Java Platform, Enterprise Edition (Java EE) resources including servlets, JavaServer Pages (JSP) files, and Enterprise JavaBeans (EJB) components.

Refer to “Java 2 security” on page 74 for more information.

The following sections cover the JAAS implementation and programming model:
- Login configuration for Java Authentication and Authorization Service
- Programmatic login for JAAS
- “Java Authentication and Authorization Service authorization”

The JAAS documentation can be found at http://www.ibm.com/developerworks/java/jdk/security. Scroll down to find the JAAS documentation for your platform.

Java Authentication and Authorization Service authorization

Java 2 security architecture uses a security policy to specify which access rights are granted to running code. This architecture is code-centric. The permissions are granted based on code characteristics

including where the code is coming from, whether it is digitally signed, and by whom. Authorization of the Java Authentication and Authorization Service (JAAS) augments the existing code-centric access controls with new user-centric access controls. Permissions are granted based on what code is running and who is running it.

When using JAAS authentication to authenticate a user, a subject is created to represent the authenticated user. A subject is comprised of a set of principals, where each principal represents an identity for that user. You can grant permissions in the policy to specific principals. After the user is authenticated, the application can associate the subject with the current access control context. For each subsequent security-checked operation, the Java runtime automatically determines whether the policy grants the required permission to a specific principal only. If so, the operation is supported if the subject that is associated with the access control context contains the designated principal only.

Associate a subject with the current access control context by calling the static doAs method from the subject class, passing it an authenticated subject and the java.security.PrivilegedAction or java.security.PrivilegedExceptionAction method. The doAs method associates the provided subject with the current access control context and then invokes the run method from the action. The run method implementation contains all the code that ran as the specified subject. The action runs as the specified subject.

In the Java 2 Platform, Enterprise Edition (J2EE) programming model, when invoking the Enterprise JavaBeans (EJB) method from an enterprise bean or servlet, the method runs under the user identity that is determined by the run-as setting. The J2EE Version 1.4 Specification does not indicate which user identity to use when invoking an enterprise bean from a Subject.doAs action block within either the EJB code or the servlet code. A logical extension is to use the proper identity that is specified in the subject when invoking the EJB method within the Subject doAs action block.

Letting the Subject.doAs action overwrite the run-as identity setting is an ideal way to integrate the JAAS programming model with the J2EE run-time environment. However, JAAS introduced an issue into the Software Development Kit (SDK), Java Technology Edition Versions 1.3 or later when integrating the JAAS Version 1.0 or later implementation with the Java 2 security architecture. A subject, which is associated with the access control context is cut off by a doPrivileged call when a doPrivileged call occurs within the Subject.doAs action block. Until this problem is corrected, no reliable and run-time efficient way is available to guarantee the correct behavior of Subject.doAs action in a J2EE run-time environment.

The problem can be explained better with the following example:

```
Subject.doAs(subject, new java.security.PrivilegedAction() {
    Public Object run() {
        // Subject is associated with the current thread context
        java.security.AccessController.doPrivileged( new
            java.security.PrivilegedAction() {
                                public Object run() {
                                // Subject was cut off from the current
                                // thread context
    return null;
                }
         });
         // Subject is associated with the current thread context
         return null;
    }
});
```

In the previous code example, the Subject object is associated with the context of the current thread. Within the run method of a doPrivileged action block, the Subject object is removed from the thread context. After leaving the doPrivileged block, the Subject object is restored to the current thread context.

Because doPrivileged blocks can be placed anywhere along the running path and instrumented quite often in a server environment, the run-time behavior of a doAs action block becomes difficult to manage.

To resolve this difficulty, WebSphere Application Server provides a WSSubject helper class to extend the JAAS authorization to a J2EE EJB method invocation, as described previously. The WSSubject class provides static doAs and doAsPrivileged methods that have identical signatures to the subject class. The WSSubject.doAs method associates the Subject to the currently running thread. The WSSubject.doAs and WSSubject.doAsPrivileged methods then invoke the corresponding Subject.doAs and Subject.doAsPrivileged methods. The original credential is restored and associated with the running thread upon leaving the WSSubject.doAs and WSSubject.doAsPrivileged methods.

The WSSubject class is not a replacement of the subject object, but rather a helper class to ensure consistent run-time behavior as long as an EJB method invocation is a concern.

The following example illustrates the run-time behavior of the WSSubject.doAs method:

```
WSSubject.doAs(subject, new java.security.PrivilegedAction() {
    Public Object run() {
        // Subject is associated with the current thread context
        java.security.AccessController.doPrivileged( new
            java.security.PrivilegedAction() {
                public Object run() {
                    // Subject was cut off from the current thread
                    // context.
    return null;
                }
         });
         // Subject is associated with the current thread context
    return null;
    }
});
```

The Subject.doAs and Subject.doAsPrivileged methods are not integrated with the J2EE run-time environment. EJB methods that are invoked within the Subject.doAs and Subject.doAsPrivileged action blocks run under the identity that is specified by the run-as setting and not by the subject identity.

- The Subject object that is generated by the WSLoginModuleImpl instance and the WSClientLoginModuleImpl instance contains a principal that implements the WSPrincipal interface. Using the getCredential method for a WSPrincipal object returns an object that implements the WSCredential interface. You can also find the WSCredential object instance in the PublicCredentials list of the subject instance. Retrieve the WSCredential object from the PublicCredentials list instead of using the getCredential method.
- The getCallerPrincipal method for the WSSubject class returns a string that represents the caller security identity. The return type differs from the getCallerPrincipal method of the java.security.Principal EJBContext interface.
- The Subject object that is generated by the Java 2 Connector (J2C) DefaultPrincipalMapping module contains a resource principal and a PasswordCredentials list. The resource principal represents the RunAs identity.

For more information, see J2EE connector security.

Using the Java Authentication and Authorization Service programming model for web authentication

WebSphere Application Server supports the Java Platform, Enterprise Edition (Java EE) declarative security model. You can define the authentication and access control policy using the Java EE deployment descriptor. You can further stack custom login modules to customize the WebSphere Application Server authentication mechanism.

Before you begin

A custom login module can perform principal and credential mapping, custom security token and custom credential-processing, and error-handling among other possibilities. Typically, you do not need to use application code to perform authentication function. Use the programming techniques that are described in this section if you have to perform authentication function in application code. Use declarative security as a rule; use the techniques that are described in this section as a last resort.

About this task

When the Lightweight Third-Party Authentication (LTPA) mechanism single sign-on (SSO) option is enabled, the web client login session is tracked by an LTPA SSO token cookie after successful login. At logout, this token is deleted to terminate the login session, but the server-side subject is not deleted. When you use the declarative security model, the WebSphere Application Server web container performs client authentication and login session management automatically. You can perform authentication in application code by setting a login page without a Java EE security constraint and by directing client requests to your login page first. Your login page can use the Java Authentication and Authorization Service (JAAS) programming model to perform authentication. To enable WebSphere Application Server web login modules to generate SSO cookies, use the following steps.

Procedure

1. Create a new system login JAAS configuration. To access the panel, click **Security > Global security**. Under Java Authentication and Authorization Service, click **System logins**.
2. Manually clone the WEB_INBOUND login configuration, and give it a new alias. To clone the login configuration, click **New**, enter a name for the configuration, click **Apply**, then click **JAAS login modules** under Additional properties. Click **New** and configure the JAAS login module. For more information, see Login module settings for Java Authentication and Authorization Service. WebSphere Application Server web container uses the WEB_INBOUND login configuration to authenticate web clients. Changing the WEB_INBOUND login configuration affects all web applications in the cell. You should create your own login configuration by cloning the contents of the WEB_INBOUND login configuration.
3. Select the `wsMapDefaultInboundLoginModule` login module and click **Custom properties**. There are two login modules defined in your login configuration: `ltpaLoginModule` and `wsMapDefaultInboundLoginModule`.
4. Add a login property name *cookie* with a value of **true**. The two login modules are enabled to generate LTPA SSO cookies. Do not add the cookie login option to the original WEB_INBOUND login configuration.
5. Stack your custom LoginModule(s) in the new login configuration (optional).
6. Use your login page for programmatic login by perform a JAAS LoginContext.login using your newly defined login configuration. After a successful login, either the `ltpaLoginModule` or the `wsMapDefaultInboundLoginModule` generates an LTPA SSO cookie upon a successful authentication. Exactly which LoginModule generates the SSO cookie depends on many factors, including system authentication configuration and runtime condition (which is beyond the scope of this section).
7. Call the modified `WSSubject.setRunAsSubject` method to add the subject to the authentication cache. The subject must be a WebSphere Application Server JAAS subject created by `LoginModule`. Adding the subject to the authentication cache recreates a subject from SSO token.

8. Use your programmatic logout page to revoke SSO cookies by invoking the `revokeSSOCookies` method from the WSSecurityHelper class.

   The term "cookies" is used because WebSphere Application Server Version 5.1.1 and later support a new LTPA SSO token with a different encryption algorithm but can be configured to generate the original LTPA SSO token for backward compatibility. Note that the subject is still in the authentication cache and only the SSO cookies are revoked.

   Note: The revokeSSOCookies(HttpServletRequest, HttpServletResponse) method from the WSSecurityHelper class is deprecated. Use the functionality provided by the Java Servlet-3.0 logout() method. Read “Servlet security methods” on page 856.

Example

Use the following code sample to perform authentication.

gotcha: If you set the password for the WSCallbackHandlerFactoryset factory class for getting handlers to `null`, as is done in the following example, you allow identity assertion without a password.

```
Suppose you wrote a LoginServlet.java:

 Import com.ibm.wsspi.security.auth.callback.WSCallbackHandlerFactory;
 Import com.ibm.websphere.security.auth.WSSubject;

 public Object login(HttpServletRequest req, HttpServletResponse res)
 throws ServletException {

 PrintWriter out = null;
 try {
  out =  res.getWriter();
       res.setContentType("text/html");
 } catch (java.io.IOException e){
  // Error handling
 }

 Subject subject = null;
 try {
 LoginContext lc = new LoginContext("system.Your_login_configuration",
WSCallbackHandlerFactory.getInstance().getCallbackHandler(
userid, null, password, req, res, null));
  lc.login();
  subject = lc.getSubject();
       WSSubject.setRunAsSubject(subject);
 } catch(Exception e) {
  // catch all possible exceptions if you want or handle them separately
  out.println("Exception in LoginContext login + Exception = " +
 e.getMessage());
  throw new ServletException(e.getMessage());
 }

The following is sample code to revoke the SSO cookies upon a programming logout:


The LogoutServlet.java:

 public void logout(HttpServletRequest req, HttpServletResponse res,
 Object retCreds)  throws ServletException {
   PrintWriter out =null;
   try {
    out =  res.getWriter();
         res.setContentType("text/html");
   } catch (java.io.IOException e){
   // Error Handling
   }
   try {
     WSSecurityHelper.revokeSSOCookies(req, res);
   } catch(Exception e) {
    // catch all possible exceptions if you want or handle them separately
    out.println("JAASLogoutServlet: logout Exception = " + e.getMessage());
    throw new ServletException(e);
   }
  }
```

What to do next

For more information on JAAS authentication, refer to Developing programmatic logins with the Java Authentication and Authorization Service. For more information on the AuthenLoginModule login module,

refer to Example: Customizing a server-side Java Authentication and Authorization Service authentication and login configuration.

Developing custom login modules for a system login configuration for JAAS

For WebSphere Application Server, multiple Java Authentication and Authorization Service (JAAS) plug-in points exist for configuring system logins. WebSphere Application Server uses system login configurations to authenticate incoming requests, outgoing requests, and internal server logins.

About this task

Application login configurations are called by Java Platform, Enterprise Edition (Java EE) applications for obtaining a Subject that is based on specific authentication information. This login configuration enables the application to associate the Subject with a specific protected remote action. The Subject is picked up on the outbound request processing. The following list identifies the main system plug-in points. If you write a login module that adds information to the Subject of a system login, these are the main login configurations to plug in:

- WEB_INBOUND
- RMI_OUTBOUND
- RMI_INBOUND
- DEFAULT

Procedure

- Authenticate web requests with the WEB_INBOUND login configuration.

  The WEB_INBOUND login configuration authenticates web requests.

  For more detailed information on the WEB_INBOUND configuration including its associated callbacks, see “RMI_INBOUND, WEB_INBOUND, DEFAULT” inSystem login configuration entry settings for Java Authentication and Authorization Service. Figure 1 shows an example of a configuration using a trust association interceptor (TAI) that creates a Subject with the initial information that is passed into the WEB_INBOUND login configuration. If the trust association interceptor is not configured, the authentication process goes directly to the WEB_INBOUND system login configuration, which consists of all the login modules combined in Figure 1. Figure 1 shows where you can plug in custom login modules and where the ltpaLoginModule and the wsMapDefaultInboundLoginModule login modules are required.

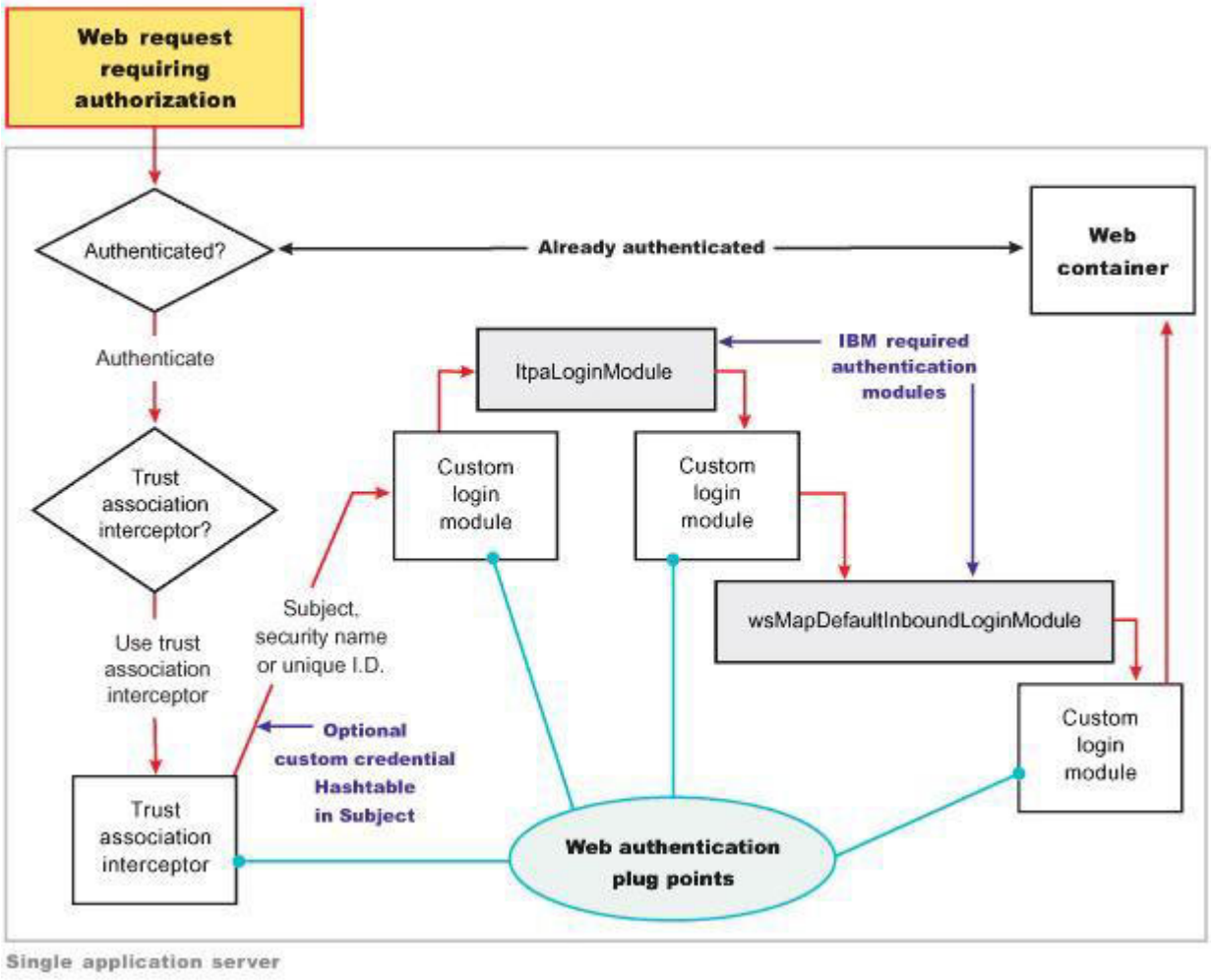


Figure 19. WEB_INBOUND login configuration

- Handle outbound requests with the RMI_OUTBOUND login configuration.

  The RMI_OUTBOUND login configuration is a plug point for handling outbound requests. WebSphere Application Server uses this plug point to create the serialized information that is sent downstream based on the invocation Subject passed in and other security context information such as propagation tokens. A custom login module can use this plug point to change the identity. For more information, see “Configuring outbound identity mapping to a different target realm” on page 464. Figure 2 shows where you can plug in custom login modules and shows where the wsMapCSIv2OutboundLoginModule login module is required.

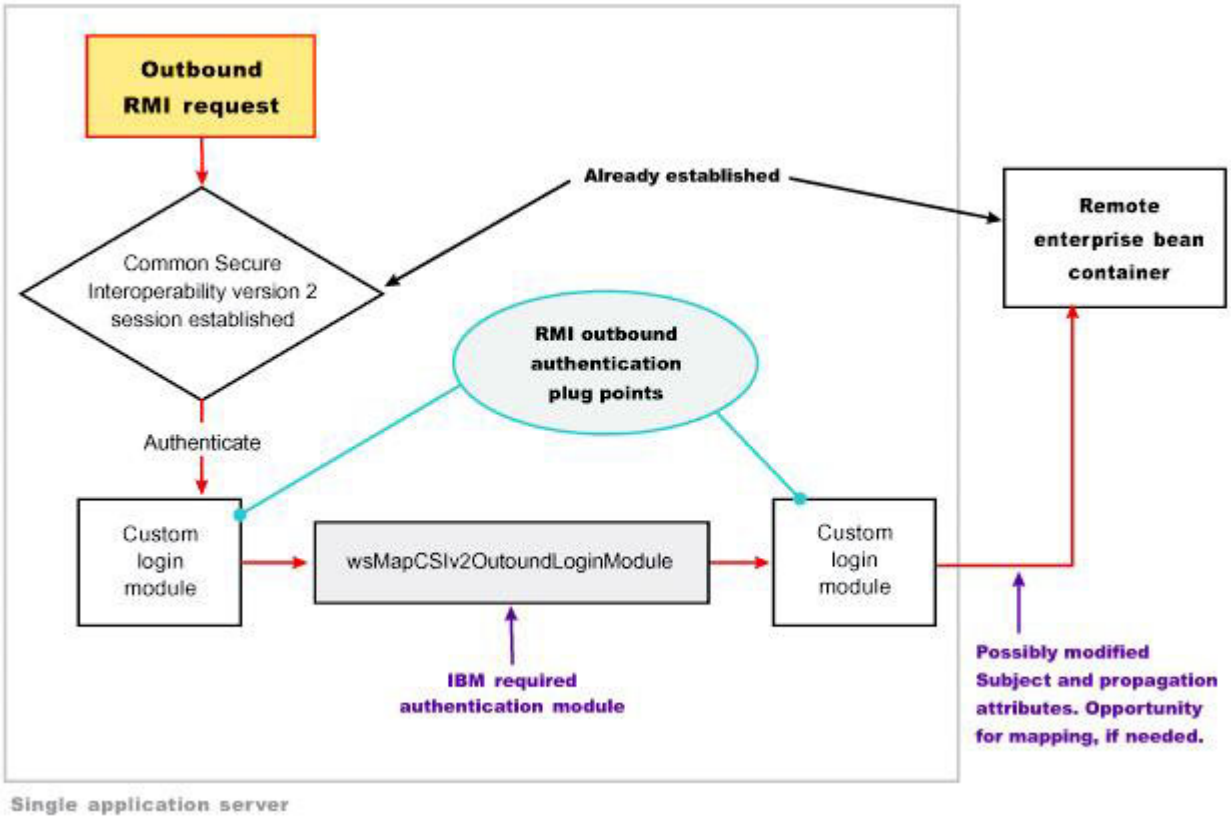


Figure 20. RMI_OUTBOUND login configuration

For more information on the RMI_OUTBOUND login configuration, including its associated callbacks, see “RMI_OUTBOUND” in System login configuration entry settings for Java Authentication and Authorization Service.

- Handle inbound authentication for enterprise bean requests with the RMI_INBOUND login configuration.

  The RMI_INBOUND login configuration is a plug point that handles inbound authentication for enterprise bean requests. WebSphere Application Server uses this plug point for either an initial login or a propagation login. For more information about these two login types, see “Security attribute propagation” on page 468. During a propagation login, this plug point is used to deserialize the information that is received from an upstream server. A custom login module can use this plug point to change the identity, handle custom tokens, add custom objects into the Subject, and so on. For more information on changing the identity using a Hashtable object, which is referenced in figure 3, see “Configuring inbound identity mapping” on page 457. Figure 3 shows where you can plug in custom login modules and shows that the ltpaLoginModule and the wsMapDefaultInboundLoginModule login modules are required.

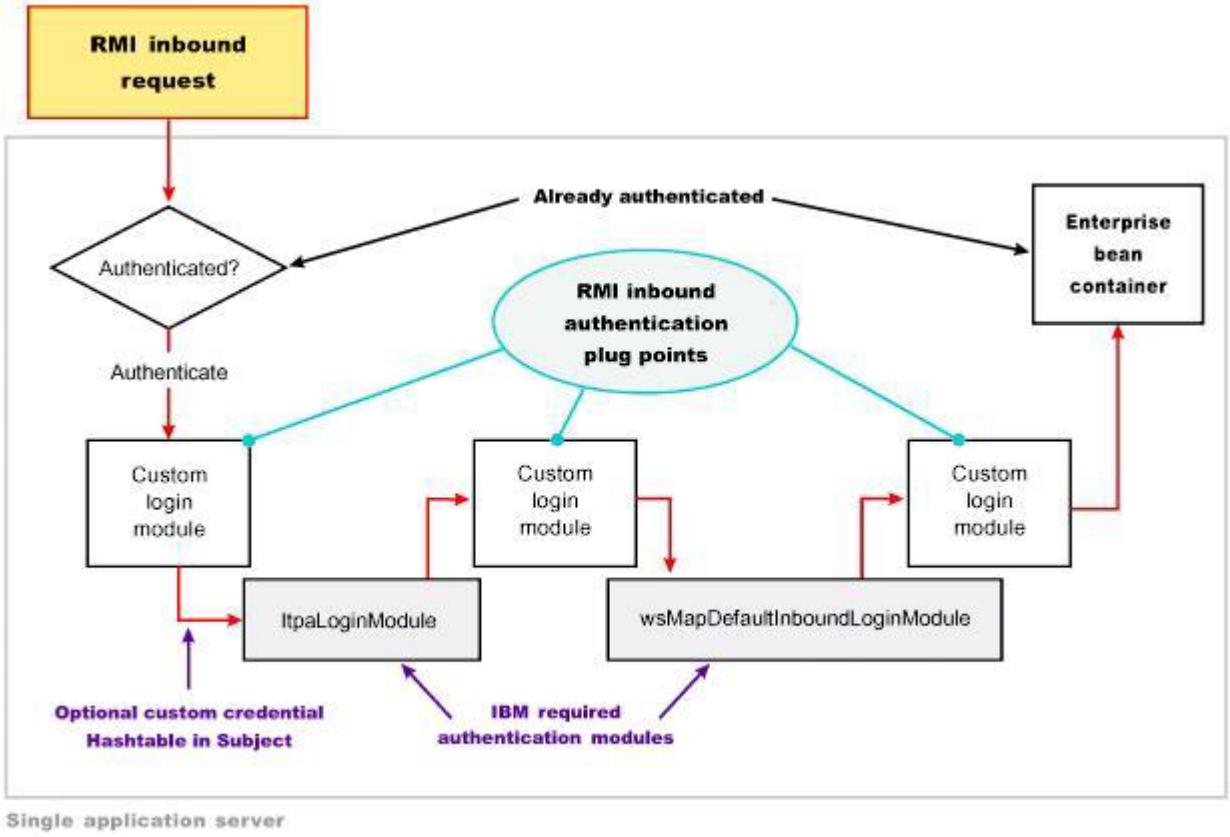


Figure 21. RMI_INBOUND login configuration

For more information on the RMI_INBOUND login configuration, including its associated callbacks, see "RMI_INBOUND, WEB_INBOUND, DEFAULT" in System login configuration entry settings for Java Authentication and Authorization Service.

- Handle all other types of authentication requests with the DEFAULT login configuration. **DEFAULT login configuration**

  The DEFAULT login configuration is a plug point that handles all of the other types of authentication requests, including administrative SOAP requests and internal authentication of the server ID. Propagation logins typically do not occur at this plug point.

  For more information on the DEFAULT login configuration including its associated callbacks, see "RMI_INBOUND, WEB_INBOUND, DEFAULT" in System login configuration entry settings for Java Authentication and Authorization Service.

- Develop login configuration logic to know when specific information is present and how to use the information. **Writing a login module**

  When you write a login module that plugs into a WebSphere Application Server application login or system login configuration, read the JAAS programming model, which is located at: http://java.sun.com/products/jaas. The JAAS programming model provides basic information about JAAS. However, before writing a login module for the WebSphere Application Server environment, read the following sections in this article:

  – Useable callbacks
  – Shared state variables
  – Initial versus propagation logins
  – Sample custom login module

  Usable Callbacks

Each login configuration must document the callbacks that are recognized by the login configuration. However, the callbacks are not always passed data. The login configuration must contain logic to know when specific information is present and how to use the information. For example, if you write a custom login module that can plug into all four of the pre-configured system login configurations mentioned previously, three sets of callbacks might be presented to authenticate a request. Other callbacks might be present for other reasons, including propagation and making other information available to the login configuration.

Login information can be presented in the following combinations:

User name (NameCallback) and password (PasswordCallback)
This information is a typical authentication combination.

User name only (NameCallback)
This information is used for identity assertion, trust association interceptor (TAI) logins, and certificate logins.

Token (WSCredTokenCallbackImpl)
This information is for Lightweight Third Party Authentication (LTPA) token validation.

Propagation token list (WSTokenHolderCallback)
This information is used for a propagation login.

The first three combinations are used for typical authentication. However, when the WSTokenHolderCallback callback is present in addition to one of the first three information combinations, the login is called a *propagation login*. A propagation login means that some security attributes are propagated to this server from another server. The servers can reuse these security attributes if the authentication information validates successfully. In some cases, a WSTokenHolderCallback callback might not have sufficient attributes for a full login. Check the requiresLogin method on the WSTokenHolderCallback callback to determine if a new login is required. You can always ignore the information returned by the requiresLogin method, but, as a result, you might duplicate information. The following list contains the callbacks that might be present in the system login configurations. The list includes the callback name and a description of their responsibility.

callbacks[0] = new javax.security.auth.callback.NameCallback(“Username: ”);
This callback handler collects the user name for the login. The result can be the user name for a basic authentication login (user name and password) or a user name for an identity assertion login.

callbacks[1] = new javax.security.auth.callback.PasswordCallback(“Password: ”, false);
This callback handler collects the password for the login.

callbacks[2] = new com.ibm.websphere.security.auth.callback.WSCredTokenCallbackImpl(“Credential Token:”);
This callback handler collects the Lightweight Third Party Authentication (LTPA) token or other token type for the login. This callback handler is typically present when a user name and password are not present.

callbacks[3] = new com.ibm.wsspi.security.auth.callback.WSTokenHolderCallback(“Authz Token List:”);
This callback handler collects the ArrayList of TokenHolder objects that are returned from a call to the WSOpaqueTokenHelper.createTokenHolderListFromOpaqueToken API using the Common Secure Interoperability Version 2 (CSIv2) authorization token as input.

callbacks[4] = new com.ibm.websphere.security.auth.callback.WSServletRequestCallback(“HttpServletRequest:”);
This callback handler collects the HTTP servlet request object, if present. This callback handler enables login modules to get information from the HTTP request for use in the login, and is presented from the WEB_INBOUND login configuration only.

callbacks[5] = new com.ibm.websphere.security.auth.callback.WSServletResponseCallback(“HttpServletResponse:”);

This callback handler collects the HTTP servlet response object, if present. This callback handler enables login modules to put information into the HTTP response as a result of the login. An example of this situation might be adding the SingleSignonCookie cookie to the response.This callback handler is presented from the WEB_INBOUND login configuration only.

callbacks[6] = new com.ibm.websphere.security.auth.callback.WSAppContextCallback("ApplicationContextCallback:");
This callback handler collects the web application context that is used during the login. This callback handler consists of a HashMap object, which contains the application name and the redirect web address, if present. The callback handler is presented from the WEB_INBOUND login configuration only.

callbacks[7] = new WSRealmNameCallbackImpl("Realm Name:", *default_realm***);**
This callback handler collects the realm name for the login information. The realm information might not always be provided. If the realm information is not provided, assume that it is the current realm.

callbacks[8] = new WSX509CertificateChainCallback("X509Certificate[]: ");
This callback handler contains the certificate that was validated by Secure Sockets Layer (SSL) if the login source is an X509Certificate from SSL client authentication. The ltpaLoginModule calls the same mapping functions as WebSphere Application Server releases prior to version 6.1. However, having it passed into the login gives a custom login module the opportunity to map the certificate in a custom way. Then, it performs a Hashtable login. See "Configuring inbound identity mapping" on page 457 for more information on a Hashtable login.

- Use shared state variables to share information between login modules during the login phase.

  If you want to access the objects that WebSphere Application Server creates during a login, refer to the following shared state variables. The variables are set in the following login modules: ltpaLoginModule, swamLoginModule, and wsMapDefaultInboundLoginModule.

  Shared state variable
  com.ibm.wsspi.security.auth.callback.Constants.WSPRINCIPAL_KEY

  Purpose
  Specifies the com.ibm.websphere.security.auth.WSPrincipal object. See the WebSphere Application Server API documentation for application programming interface (API) usage. This shared state variable is for read-only purposes. Do not set this variable in the shared state for custom login modules.

  The login module in which variables are set
  ltpaLoginModule, swamLoginModule, and wsMapDefaultInboundLoginModule

  Shared state variable
  com.ibm.wsspi.security.auth.callback.Constants.WSCREDENTIAL_KEY

  Purpose
  Specifies the com.ibm.websphere.security.cred.WSCredential object. See the WebSphere Application Server API documentation for API usage. This shared state variable is for read-only purposes. Do not set this variable in the shared state for custom login modules.

  Login module in which variables are set
  wsMapDefaultInboundLoginModule

  Shared state variable
  com.ibm.wsspi.security.auth.callback.Constants.WSAUTHZTOKEN_KEY

  Purpose
  Specifies the default com.ibm.wsspi.security.token.AuthorizationToken object. Login modules can use this object to set custom attributes plugged in after the wsMapDefaultInboundLoginModule login module. The information set here is propagated downstream and is available to the application. See the WebSphere Application Server API documentation for API usage.

Initial versus propagation logins

As mentioned previously, some logins are considered initial logins because of the following reasons:

– It is the first time authentication information is presented to WebSphere Application Server.
– The login information is received from a server that does not propagate security attributes so this information must be gathered from a user registry.

Other logins are considered propagation logins when a WSTokenHolderCallback callback is present and contains sufficient information from a sending server to recreate all the required objects needed by WebSphere Application Server runtime. In cases where there is sufficient information for the WebSphere Application Server runtime, the information you might add to the Subject is likely to exist from the previous login. To verify if your object is present, you can get access to the ArrayList object that is present in the WSTokenHolderCallback callback, and search through this list looking at each TokenHolder getName method. This search is used to determine if WebSphere Application Server is deserializing your custom object during this login. Check the class name returned from the getName method using the String startsWith method because the runtime might add additional information at the end of the name to know which Subject is set to add the custom object after deserialization.

- Code your login() method to determine when sufficient information is present.

  The following code snippet can be used in your login() method to determine when sufficient information is present. For another example, see “Configuring inbound identity mapping” on page 457.

```
// This is a hint provided by WebSphere Application Server that
// sufficient propagation information does not exist and, therefore,
// a login is required to provide the sufficient information. In this
// situation, a Hashtable login might be used.
boolean requiresLogin = ((com.ibm.wsspi.security.auth.callback.
WSTokenHolderCallback) callbacks[1]).requiresLogin();

if (requiresLogin)
{
// Check to see if your object exists in the TokenHolder list,
if not, add it.
java.util.ArrayList authzTokenList = ((WSTokenHolderCallback) callbacks[6]).
getTokenHolderList();boolean found = false;

if (authzTokenList != null)
{
Iterator tokenListIterator = authzTokenList.iterator();

while (tokenListIterator.hasNext())
{
com.ibm.wsspi.security.token.TokenHolder th = (com.ibm.wsspi.security.token.
TokenHolder) tokenListIterator.next();

if (th != null && th.getName().startsWith("com.acme.myCustomClass"))
{
found=true;
break;
}
}
if (!found)
{
// go ahead and add your custom object.
}
}
}
else
{
// This code indicates that sufficient propagation information is present.
// User registry calls are not needed by WebSphere Application Server to
// create a valid Subject. This code might be a no-op in your login module.
}
```

Sample custom login module

You can use the following sample to get ideas on how to use some of the callbacks and shared state variables.

```
{
 // Defines your login module variables
 com.ibm.wsspi.security.token.AuthenticationToken customAuthzToken = null;
 com.ibm.wsspi.security.token.AuthenticationToken defaultAuthzToken = null;
 com.ibm.websphere.security.cred.WSCredential credential = null;
 com.ibm.websphere.security.auth.WSPrincipal principal = null;
 private javax.security.auth.Subject _subject;
 private javax.security.auth.callback.CallbackHandler _callbackHandler;
 private java.util.Map _sharedState;
 private java.util.Map _options;
```

```
public void initialize(Subject subject, CallbackHandler callbackHandler,
    Map sharedState, Map options)
{
 _subject = subject;
 _callbackHandler = callbackHandler;
 _sharedState = sharedState;
 _options = options;
}

public boolean login() throws LoginException
{
 boolean succeeded = true;

 // Gets the CALLBACK information
 javax.security.auth.callback.Callback callbacks[] = new javax.security.
       auth.callback.Callback[7];
 callbacks[0] = new javax.security.auth.callback.NameCallback(
       "Username: ");
 callbacks[1] = new javax.security.auth.callback.PasswordCallback(
       "Password: ", false);
 callbacks[2] = new com.ibm.websphere.security.auth.callback.
       WSCredTokenCallbackImpl ("Credential Token: ");
 callbacks[3] = new com.ibm.wsspi.security.auth.callback.
       WSServletRequestCallback ("HttpServletRequest: ");
 callbacks[4] = new com.ibm.wsspi.security.auth.callback.
       WSServletResponseCallback ("HttpServletResponse: ");
 callbacks[5] = new com.ibm.wsspi.security.auth.callback.
       WSAppContextCallback ("ApplicationContextCallback: ");
 callbacks[6] = new com.ibm.wsspi.security.auth.callback.
        WSTokenHolderCallback ("Authz Token List: ");

 try
 {
  callbackHandler.handle(callbacks);
 }
 catch (Exception e)
 {
  // Handles exceptions
  throw new WSLoginFailedException (e.getMessage(), e);
 }

 // Sees which callbacks contain information
 uid = ((NameCallback) callbacks[0]).getName();
 char password[] = ((PasswordCallback) callbacks[1]).getPassword();
 byte[] credToken = ((WSCredTokenCallbackImpl) callbacks[2]).getCredToken();
 javax.servlet.http.HttpServletRequest request = ((WSServletRequestCallback)
       callbacks[3]).getHttpServletRequest();
 javax.servlet.http.HttpServletResponse response = ((WSServletResponseCallback)
       callbacks[4]).getHttpServletResponse();
 java.util.Map appContext = ((WSAppContextCallback)
       callbacks[5]).getContext();
 java.util.List authzTokenList = ((WSTokenHolderCallback)
       callbacks[6]).getTokenHolderList();

 // Gets the SHARED STATE information
 principal = (WSPrincipal) _sharedState.get(com.ibm.wsspi.security.
       auth.callback.Constants.WSPRINCIPAL_KEY);
 credential = (WSCredential) _sharedState.get(com.ibm.wsspi.security.
       auth.callback.Constants.WSCREDENTIAL_KEY);
 defaultAuthzToken = (AuthorizationToken) _sharedState.get(com.ibm.
       wsspi.security.auth.callback.Constants.WSAUTHZTOKEN_KEY);

    // What you tend to do with this information depends upon the scenario
    // that you are trying to accomplish. This example demonstrates how to
    // access various different information:
    // - Determine if a login is initial versus propagation
    // - Deserialize a custom authorization token (For more information, see
    //   "Security attribute propagation" on page 468
    // - Add a new custom authorization token (For more information, see
    //   "Security attribute propagation" on page 468
    // - Look for a WSCredential and read attributes, if found.
    // - Look for a WSPrincipal and read attributes, if found.
    // - Look for a default AuthorizationToken and add attributes, if found.
    // - Read the header attributes from the HttpServletRequest, if found.
    // - Add an attribute to the HttpServletResponse, if found.
    // - Get the web application name from the appContext, if found.

    // - Determines if a login is initial versus propagation. This is most
    //  useful when login module is first.
 boolean requiresLogin = ((WSTokenHolderCallback) callbacks[6]).requiresLogin();

 // initial login - asserts privilege attributes based on user identity
 if (requiresLogin)
 {

  // If you are validating a token from another server, there is an
      // application programming interface  (API) to get the uniqueID from it.
  if (credToken != null && uid == null)
  {
   try
```

```
 {
  String uniqueID = WSSecurityPropagationHelper.
            validateLTPAToken(credToken);
  String realm = WSSecurityPropagationHelper.getRealmFromUniqueID
             (uniqueID);
         // Now set it to the UID so you can use that to either map or
         // login with.
  uid = WSSecurityPropagationHelper.getUserFromUniqueID (uniqueID);
 }
 catch (Exception e)
 {
  // handle exception
 }
}
    // Adds a Hashtable to shared state.
    // Note: You can perform custom mapping on the NameCallback value returned
    // to change the identity based upon your own mapping rules.
uid = mapUser (uid);

// Gets the default InitialContext for this server.
javax.naming.InitialContext ctx = new javax.naming.InitialContext();

// Gets the local UserRegistry object.
com.ibm.websphere.security.UserRegistry reg = (com.ibm.websphere.security.
         UserRegistry) ctx.lookup("UserRegistry");

    // Gets the user registry uniqueID based on the uid specified in the
    // NameCallback.
String uniqueid = reg.getUniqueUserId(uid);
 uid = WSSecurityPropagationHelper.getUserFromUniqueID (uniqueID);

// Gets the display name from the user registry based on the uniqueID.
String securityName = reg.getUserSecurityName(uid);

// Gets the groups associated with this uniqueID.
java.util.List groupList = reg.getUniqueGroupIds(uid);

    // Creates the java.util.Hashtable with the information you gathered from
    // the UserRegistry.
java.util.Hashtable hashtable = new java.util.Hashtable();
hashtable.put(com.ibm.wsspi.security.token.AttributeNameConstants.
        WSCREDENTIAL_UNIQUEID, uniqueid);
hashtable.put(com.ibm.wsspi.security.token.AttributeNameConstants.
        WSCREDENTIAL_SECURITYNAME, securityName);
hashtable.put(com.ibm.wsspi.security.token.AttributeNameConstants.
        WSCREDENTIAL_GROUPS, groupList);

    // Adds a cache key that is used as part of the lookup mechanism for
    // the created Subject. The cache key can be an Object, but should
    // implement the toString() method. Make sure the cacheKey contains
    // enough information to scope it to the user and any additional
    // attributes that you use. If you do not specify this property the
    // Subject is scoped to the WSCREDENTIAL_UNIQUEID returned, by default.
hashtable.put(com.ibm.wsspi.security.token.AttributeNameConstants.
       WSCREDENTIAL_CACHE_KEY,
  "myCustomAttribute" + uniqueid);

 // Adds the hashtable to the sharedState of the Subject.
 _sharedState.put(com.ibm.wsspi.security.token.AttributeNameConstants.
          WSCREDENTIAL_PROPERTIES_KEY,hashtable);
}
// propagation login - process propagated tokens
else
{
// - Deserializes a custom authorization token. For more information, see
    //    "Security attribute propagation" on page 468.
    //    This can be done at any login module plug in point (first,
    //     middle, or last).
 if (authzTokenList != null)
 {
  // Iterates through the list looking for your custom token
  for (int i=0; i<authzTokenList.size(); i++)
  {
   TokenHolder tokenHolder = (TokenHolder)authzTokenList.get(i);

   // Looks for the name and version of your custom AuthorizationToken
          // implementation
   if (tokenHolder.getName().equals("com.ibm.websphere.security.token.
            CustomAuthorizationTokenImpl") && tokenHolder.getVersion() == 1)
   {
    // Passes the bytes into your custom AuthorizationToken constructor
           // to deserialize
    customAuthzToken = new
     com.ibm.websphere.security.token.
                CustomAuthorizationTokenImpl(tokenHolder.getBytes());

   }
  }
 }
     // - Adds a new custom authorization token (For more information,
```

```
      //   see "Security attribute propagation" on page 468)
      //    This can be done at any login module plug in point (first, middle,
      //    or last).
 else
 {
  // Gets the PRINCIPAL from the default AuthenticationToken. This must
        // match all of the tokens.
  defaultAuthToken = (com.ibm.wsspi.security.token.AuthenticationToken)
   sharedState.get(com.ibm.wsspi.security.auth.callback.Constants.
             WSAUTHTOKEN_KEY);
  String principal = defaultAuthToken.getPrincipal();

  // Adds a new custom authorization token. This is an initial login.
        // Pass the principal into the constructor
  customAuthzToken = new com.ibm.websphere.security.token.
             CustomAuthorizationTokenImpl(principal);

  // Adds any initial attributes
  if (customAuthzToken != null)
  {
   customAuthzToken.addAttribute("key1", "value1");
   customAuthzToken.addAttribute("key1", "value2");
   customAuthzToken.addAttribute("key2", "value1");
   customAuthzToken.addAttribute("key3", "something different");
  }
 }
}

// - Looks for a WSCredential and read attributes, if found.
//   This is most useful when plugged in as the last login module.
if (credential != null)
{
 try
 {
  // Reads some data from the credential
  String securityName = credential.getSecurityName();
  java.util.ArrayList = credential.getGroupIds();
 }
 catch (Exception e)
 {
  // Handles exceptions
  throw new WSLoginFailedException (e.getMessage(), e);
 }
}

// - Looks for a WSPrincipal and read attributes, if found.
//   This is most useful when plugged as the last login module.
if (principal != null)
{
 try
 {
  // Reads some data from the principal
  String principalName = principal.getName();
 }
 catch (Exception e)
 {
  // Handles exceptions
  throw new WSLoginFailedException (e.getMessage(), e);
 }
}

// - Looks for a default AuthorizationToken and add attributes, if found.
//   This is most useful when plugged in as the last login module.
if (defaultAuthzToken != null)
{
 try
 {
  // Reads some data from the defaultAuthzToken
  String[] myCustomValue = defaultAuthzToken.getAttributes ("myKey");
  // Adds some data if not present in the defaultAuthzToken
  if (myCustomValue == null)
   defaultAuthzToken.addAttribute ("myKey", "myCustomData");
 }
 catch (Exception e)
 {
  // Handles exceptions
  throw new WSLoginFailedException (e.getMessage(), e);
 }
}

// - Reads the header attributes from the HttpServletRequest, if found.
//  This can be done at any login module plug in point (first, middle,
   //  or last).
if (request != null)
{
 java.util.Enumeration headerEnum = request.getHeaders();
 while (headerEnum.hasMoreElements())
 {
  System.out.println ("Header element: " + (String)headerEnum.nextElement());
 }
```

```
 }

 // - Adds an attribute to the HttpServletResponse, if found
    //  This can be done at any login module plug in point (first, middle,
    //  or last).
 if (response != null)
 {
  response.addHeader ("myKey", "myValue");
 }

 // - Gets the web application name from the appContext, if found
 //   This can be done at any login module plug in point (first, middle,
    //   or last).
 if (appContext != null)
 {
  String appName = (String) appContext.get(com.ibm.wsspi.security.auth.
         callback.Constants.WEB_APP_NAME);
 }

 return succeeded;
}

public boolean commit() throws LoginException
{
 boolean succeeded = true;

 // Add any objects here that you have created and belong in the
 // Subject. Make sure the objects are not already added. If you added
 // any sharedState variables, remove them before you exit. If the abort()
    // method gets called, make sure you cleanup anything added to the
 // Subject here.

 if (customAuthzToken != null)
 {
  // Sets the customAuthzToken token into the Subject
  try
  {
   // Do this in a doPrivileged code block so that application code
         // does not need to add additional permissions
   java.security.AccessController.doPrivileged(new java.security.PrivilegedAction()
   {
    public Object run()
    {
     try
     {
      // Adds the custom authorization token if it is not
                //  null and not already in the Subject
                              if ((customAuthzTokenPriv != null) &&
        (!_subject.getPrivateCredentials().contains(customAuthzTokenPriv)))
      {
       _subject.getPrivateCredentials().add(customAuthzTokenPriv);
      }
     }
     catch (Exception e)
     {
      throw new WSLoginFailedException (e.getMessage(), e);
     }

     return null;
    }
   });
  }
  catch (Exception e)
  {
   throw new WSLoginFailedException (e.getMessage(), e);
  }
 }

 return succeeded;
}

public boolean abort() throws LoginException
{
 boolean succeeded = true;

 // Makes sure to remove all objects that have already been added (both into the
    // Subject and shared state).

 if (customAuthzToken != null)
 {
  // remove the customAuthzToken token from the Subject
  try
  {
   final AuthorizationToken customAuthzTokenPriv = customAuthzToken;
   // Do this in a doPrivileged block so that application code does not need
         // to add additional permissions
   java.security.AccessController.doPrivileged(new java.security.PrivilegedAction()
   {
    public Object run()
    {
```

```
      try
      {
       // Removes the custom authorization token if it is not
                  // null and not already in the Subject
                      if ((customAuthzTokenPriv != null) &&
                       (_subject.getPrivateCredentials().
                        contains(customAuthzTokenPriv)))
       {
        _subject.getPrivateCredentials().
                     remove(customAuthzTokenPriv);
       }
      }
      catch (Exception e)
      {
       throw new WSLoginFailedException (e.getMessage(), e);
      }

      return null;
     }
    });
   }
   catch (Exception e)
   {
    throw new WSLoginFailedException (e.getMessage(), e);
   }
  }

  return succeeded;
 }


 public boolean logout() throws LoginException
 {
  boolean succeeded = true;

  // Makes sure to remove all objects that have already been added
     // (both into the Subject and shared state).

  if (customAuthzToken != null)
  {
   // Removes the customAuthzToken token from the Subject
   try
   {
    final AuthorizationToken customAuthzTokenPriv = customAuthzToken;
    // Do this in a doPrivileged code block so that application code does
          // not need to add additional permissions
    java.security.AccessController.doPrivileged(new java.security.
            PrivilegedAction()
    {
     public Object run()
     {
      try
      {
       // Removes the custom authorization token if it is not null and not
                  //  already in the Subject
                      if ((customAuthzTokenPriv != null) && (_subject.
                      getPrivateCredentials().
                      contains(customAuthzTokenPriv)))
       {
        _subject.getPrivateCredentials().remove(customAuthzTokenPriv);
       }
      }
      catch (Exception e)
      {
       throw new WSLoginFailedException (e.getMessage(), e);
      }

      return null;
     }
    });
   }
   catch (Exception e)
   {
    throw new WSLoginFailedException (e.getMessage(), e);
   }
  }

  return succeeded;

 }

}
```

- Configure the system login for your custom login module.

  After developing your custom login module for a system login configuration, you can configure the system login using either the administrative console or using the wsadmin utility. To configure the system login using the administrative console, click **Security > Global security**. Under Java

Authentication and Authorization Service, click **System logins**. For more information on using the wsadmin utility for system login configuration, see Customizing a server-side Java Authentication and Authorization Service authentication and login configuration. Also refer to that article for information on system login modules and to determine whether to add additional login modules.

Customizing application login with Java Authentication and Authorization Service

Using Java Authentication and Authorization Service (JAAS), you can customize your application login.

About this task

Java Authentication and Authorization Service (JAAS) is an API that enables applications to access authentication and access control services without being tied to those services. The following topics explaining customizing your application with JAAS are covered in this section:

Procedure

1. Develop programmatic logins with JAAS.

   You can develop programmatic logins with JAAS, which represents the strategic application programming interfaces (API) for authentication.
2. Configure programmatic logins with JAAS.

   A new JAAS login configuration can be added and modified using the administrative console. The changes are saved in the cell-level security document and are available to all managed application servers.
3. Customize an application login to perform an identity assertion using JAAS.

   Using the JAAS login framework, you can create a JAAS login configuration that can be used to perform login to an identity assertion.
4. Configure a server-side JAAS authentication and login configuration.

   WebSphere Application Server supports plugging in a custom JAAS login module before or after the WebSphere Application Server system login module. However, WebSphere Application Server does not support the replacement of the WebSphere Application Server system login modules, which are used to create the WSCredential credential and WSPrincipal principal in the Subject. By using a custom login module, you can either make additional authentication decisions or add information to the Subject to make additional, potentially finer-grained, authorization decisions inside a Java Platform, Enterprise Edition (Java EE) application.

Enabling identity assertion with trust validation using JAAS:

By enabling identity assertion with trust validation, an application can use the JAAS login configuration to perform a programmatic identity assertion.

About this task

To enable an identity assertion with trust validation, follow these steps:

Procedure

1. Create a custom login module to perform a trust validation. The login module must set trust and identity information in the shared state, which is then passed on to the IdentityAssertionLoginModule. The trust and identity information is stored in a map in the shared state under the key, com.ibm.wsspi.security.common.auth.module.IdentityAssertionLoginModule.state. If this key is missing from the shared state, a WSLoginFailedException error is thrown by the IdentityAssertionLoginModule module. The custom login module should include the following:
   - A trust key named com.ibm.wsspi.security.common.auth.module.IdentityAssertionLoginModule.trust. If the trust key is set to `true`, trust is established. If the trust key is set to `false`, the IdentityAssertionLoginModule module creates a WSLoginFailedException error.

- The identity of the java.security.Principal type set in the com.ibm.wsspi.security.common.auth.module.IdenityAssertionLoginModule.principal key.
- The identity in the form of a java,security.cert.X509Certificate[] certificate set in the com.ibm.wsspi.security.common.auth.module.IdentityAssertionLoginModule.certificates key.

Note: If both a principal and a certificate are supplied, the principal is used, and a warning is issued.

2. Create a new Java Authentication and Authorization Service (JAAS) configuration for application logins. It contains the user-implemented trust validation custom login module and the IdentityAssertionLoginModule module. To configure an application login configuration from the administrative console, complete the following steps:
   a. Click **Security > Global security**.
   b. Under Java Authentication and Authorization Service, click **Application logins > New**.
   c. Supply the JAAS configuration with an alias, and then click **Apply**.
   d. Under Additional properties, click **JAAS Login Modules > New**.
   e. Enter the module class name of the user-implemented trust validation custom login module, and then click **Apply**.
   f. Enter the com.ibm.wsspi.security.common.auth.module.IdentityAssertionLoginModule module class name.
   g. Make sure that the module class name classes are in the correct order. The user-implemented trust validation login module must be the first class in the list, and the IdentityAssertionLoginModule module must be the second class.
   h. Click **Save**. The new JAAS configuration is used by the application to perform an identity assertion.

What to do next

An application can now use the JAAS login configuration to perform a programmatic identity assertion. The application can create a login context for the JAAS configuration created in step 2, then login to that login context with the identity it asserts to. If the login is successful, that identity can be set in the current running process, as in the following example:

```
MyCallbackHandler handler = new MyCallbackHandler(new MyPrincipal("Joe"));
LoginContext lc = new LoginContext("MyAppLoginConfig", handler);
lc.login();  //assume successful
Subject s = lc.getSubject();
WSSubject.setRunAsSubject(s);
// From here on, the runas identity is "Joe"
```

Performing identity mapping for authorization across servers in different realms

Identity mapping is a one-to-one mapping of a user identity between two servers so that the proper authorization decisions are made by downstream servers. Identity mapping is necessary when the integration of servers is needed, but the user registries are different and not shared between the systems.

About this task

In most cases, requests flow downstream between two servers that are part of the same security domain. In WebSphere Application Server, two servers that are members of the same cell are also members of the same security domain. In the same cell, the two servers have the same user registry and the same Lightweight Third Party Authentication (LTPA) keys for token encryption. These two commonalities ensure that the LTPA token, among other user attributes, which flows between the two servers, not only can be decrypted and validated, but also the user identity in the token can be mapped to attributes that are recognized by the authorization engine.

The most reliable and recommended configuration involves two servers within the same cell. However, sometimes you need to integrate multiple systems that cannot use the same user registry. When the user registries are different between two servers, the security domain or realm of the target server does not match the security domain of the sending server.

WebSphere Application Server enables mapping to occur either before sending the request outbound or before enabling the existing security credentials to flow to the target server. The credentials are mapped inbound with the specification that the target realm is trusted.

An alternative to mapping is to send the user identity without the token or the password to a target server without actually mapping the identity. The use of the user identity is based on trust between the two servers. Use Common Secure Interoperability Version 2 (CSIv2) identity assertion. When enabled, the server sends just the X.509 certificate, principal name, or distinguished name (DN) based upon what was used by the original client to perform the initial authentication. During CSIv2 identity assertion, trust is established between WebSphere Application Servers.

The user identity must exist in the target user registry for identity assertion to work. This process can also enable interoperability between other Java 2 Platform, Enterprise Edition (J2EE) Version 1.4 and higher compliant application servers. If both the sending server and target servers have identity assertion configured, WebSphere Application Server always uses this method of authentication, even when both servers are in the same security domain. For more information on CSIv2 identity assertion, see “Identity assertion to the downstream server” on page 523.

When the user identity is not present in the user registry of the target server, identity mapping must occur either before the request is sent outbound or when the request comes inbound. This decision depends upon your environment and requirements. However, it is typically easier to map the user identity before the request is sent outbound for the following reasons:

- You know the user identity of the existing credential as it comes from the user registry of the sending server.
- You do not have to worry about sharing Lightweight Third Party Authentication (LTPA) keys with the other target realm because you are not mapping the identity to LTPA credentials. Typically, you are mapping the identity to a user ID and password that are present in the user registry of the target realm.

When you do perform outbound mapping, in most cases, it is recommended that you use Secure Sockets Layer (SSL) to protect the integrity and confidentiality of the security information sent across the network. If LTPA keys are not shared between servers, an LTPA token cannot be validated at the inbound server. In this case, outbound mapping is necessary because the user identity cannot be determined at the inbound server to do inbound mapping. For more information, see “Configuring outbound identity mapping to a different target realm” on page 464.

When you need inbound mapping, potentially due to the mapping capabilities of the inbound server, you must ensure that both servers have the same LTPA keys so that you can get access to the user identity. Typically, in secure communications between servers, an LTPA token is passed into the WSCredTokenCallback callback of the inbound JAAS login configuration for the purposes of client authentication. A method is available that enables you to open the LTPA token, if valid, and get access to the user unique ID so that mapping can be performed. For more information, see “Configuring inbound identity mapping” on page 457. In other cases, such as identity assertion, you might receive a user name in the NameCallback callback of the inbound login configuration that enables you to map the identity.

The following topics are covered in this section:

Procedure

- Configuring inbound identity mapping For inbound identity mapping, you can write a custom login module and configure WebSphere Application Server to run the login module first within the system login configurations. Consider the following steps when you write your custom login module: “Configuring inbound identity mapping” on page 457.

- Configuring outbound identity mapping to a different target realm By default, when WebSphere Application Server makes an outbound request from one server to another server in a different security realm, the request is rejected. This topic details alternatives for enabling one server to send outbound requests to a target server in a different realm. For more information, see “Configuring outbound identity mapping to a different target realm” on page 464

Configuring inbound identity mapping

For inbound identity mapping, write a custom login module and configure WebSphere Application Server to run the login module first within the system login configurations. Consider the following steps when you write your custom login module.

Procedure

1. Get the inbound user identity from the callbacks and map the identity, if necessary This step occurs in the login method of the login module. A valid authentication has either or both NameCallback and the WSCredTokenCallback callbacks present. The following code sample shows you how to determine the user identity:

```
javax.security.auth.callback.Callback callbacks[] =
   new javax.security.auth.callback.Callback[3];
 callbacks[0] = new javax.security.auth.callback.NameCallback("");
 callbacks[1] = new javax.security.auth.callback.PasswordCallback
     ("Password: ", false);
 callbacks[2] = new com.ibm.websphere.security.auth.callback.
     WSCredTokenCallbackImpl("");
 callbacks[3] = new com.ibm.wsspi.security.auth.callback.
     WSTokenHolderCallback("");

 try
 {
  callbackHandler.handle(callbacks);
 }
 catch (Exception e)
 {
  // Handles exceptions
  throw new WSLoginFailedException (e.getMessage(), e);
 }

 // Shows which callbacks contain information
 boolean identitySwitched = false;
 String uid = ((NameCallback) callbacks[0]).getName();
 char password[] = ((PasswordCallback) callbacks[1]).getPassword();
 byte[] credToken = ((WSCredTokenCallbackImpl) callbacks[2]).getCredToken();
 java.util.List authzTokenList = ((WSTokenHolderCallback)
     callbacks[3]).getTokenHolderList();

 if (credToken != null)
 {
  try
  {
   String uniqueID = WSSecurityPropagationHelper.validateLTPAToken(credToken);
   String realm = WSSecurityPropagationHelper.getRealmFromUniqueID (uniqueID);
       // Now set the string to the UID so that you can use the result for either
       // mapping or logging in.
   uid = WSSecurityPropagationHelper.getUserFromUniqueID (uniqueID);
  }
  catch (Exception e)
  {
   // Handles the exception
  }
 }
 else if (uid == null)
 {
     // Throws an exception if authentication data is not valid.
     // You must have either UID or CredToken
```

```
 throw new WSLoginFailedException("invalid authentication data.");
}
else if (uid != null && password != null)
{
    // This is a typical authentication. You can choose to map this ID to
    // another ID or you can skip it and allow WebSphere Application Server
    // to log in for you. When passwords are presented, be very careful to not
    // validate the password because this is the initial authentication.

 return true;
}

   // If desired, map this uid to something else and set the identitySwitched
   // boolean. If the identity was changed, clear the propagated attributes
   // below so they are not used incorrectly.
uid = myCustomMappingRoutine (uid);

   // Clear the propagated attributes because they are no longer applicable
   // to the new identity
if (identitySwitched)
{
 ((WSTokenHolderCallback) callbacks[3]).setTokenHolderList(null);
}
```

2. Check to see if attribute propagation occurred and if the attributes for the user are already present when the identity remains the same. Check to see if the user attributes are already present from the sending server to avoid duplicate calls to the user registry lookup. To check for the user attributes, use a method on the WSTokenHolderCallback callback that analyzes the information present in the callback to determine if the information is sufficient for WebSphere Application Server to create a Subject. The following code sample checks for the user attributes:

   ```
   boolean requiresLogin =
   ((com.ibm.wsspi.security.auth.callback.WSTokenHolderCallback)
   callbacks[2]).getrequiresLogin();
   ```

   If sufficient attributes are not present to form the WSCredential and the WSPrincipal objects that are needed to perform authorization, the previous code sample returns a `true` result. When the result is `false`, you can choose to discontinue processing as the necessary information exists to create the Subject without performing additional remote user registry calls.

3. Optional: Look up the required attributes from the user registry, put the attributes in a hashtable, and add the hashtable to the shared state. If the identity is switched in this login module, you must complete the following steps:

   a. Create the hashtable of attributes, as shown in the following example.

   b. Add the hashtable to the shared state.

   If the identity is not switched, but the value of the requiresLogin code sample shown previously is `true`, you can create the hashtable of attributes. However, you are not required to create a hashtable in this situation as WebSphere Application Server handles the login for you. However, you might consider creating a hashtable to gather attributes in special cases where you are using your own special user registry. Creating a UserRegistry implementation, using a hashtable, and letting WebSphere Application Server gather the user attributes for you might be the easiest solution. The following table shows how to create a hashtable of user attributes:

   ```
   if (requiresLogin || identitySwitched)
    {
     // Retrieves the default InitialContext for this server.
     javax.naming.InitialContext ctx = new javax.naming.InitialContext();

     // Retrieves the local UserRegistry implementation.
     com.ibm.websphere.security.UserRegistry reg = (com.ibm.websphere.
           security.UserRegistry)
     ctx.lookup("UserRegistry");

        // Retrieves the user registry uniqueID based on the uid specified
   ```

```
    // in the NameCallback.
  String uniqueid = reg.getUniqueUserId(uid);
   uid = WSSecurityPropagationHelper.getUserFromUniqueID (uniqueid);

    // Retrieves the display name from the user registry based on the uniqueID.
  String securityName = reg.getUserSecurityName(uid);

    // Retrieves the groups associated with the uniqueID.
  java.util.List groupList = reg.getUniqueGroupIds(uid);

    // Creates the java.util.Hashtable with the information that you gathered
    // from the UserRegistry implementation.
  java.util.Hashtable hashtable = new java.util.Hashtable();
  hashtable.put(com.ibm.wsspi.security.token.AttributeNameConstants.
       WSCREDENTIAL_UNIQUEID, uniqueid);
    hashtable.put(com.ibm.wsspi.security.token.AttributeNameConstants.
      WSCREDENTIAL_SECURITYNAME, securityName);
  hashtable.put(com.ibm.wsspi.security.token.AttributeNameConstants.
       WSCREDENTIAL_GROUPS, groupList);

    // Adds a cache key that is used as part of the lookup mechanism for
    // the created Subject. The cache key can be an object, but should have
    // an implemented toString method. Make sure that the cacheKey contains
    // enough information to scope it to the user and any additional attributes
    // that you are using. If you do not specify this property the Subject is
    // scoped to the returned WSCREDENTIAL_UNIQUEID, by default.
  hashtable.put(com.ibm.wsspi.security.token.AttributeNameConstants.
        WSCREDENTIAL_CACHE_KEY, "myCustomAttribute" + uniqueid);
  // Adds the hashtable to the sharedState of the Subject.
  _sharedState.put(com.ibm.wsspi.security.token.AttributeNameConstants.
        WSCREDENTIAL_PROPERTIES_KEY, hashtable);
 }
```

The following rules define in more detail how a hashtable login is performed. You must use a java.util.Hashtable object in either the Subject (public or private credential set) or the shared-state HashMap. The com.ibm.wsspi.security.token.AttributeNameConstants class defines the keys that contain the user information. If the Hashtable object is put into the shared state of the login context using a custom login module that is listed prior to the Lightweight Third Party Authentication (LTPA) login module, the value of the java.util.Hashtable object is searched using the following key within the shared-state hashMap:

Property
com.ibm.wsspi.security.cred.propertiesObject

Reference to the property
AttributeNameConstants.WSCREDENTIAL_PROPERTIES_KEY

Explanation
This key searches for the Hashtable object that contains the required properties in the shared state of the login context.

Expected result
A java.util.Hashtable object.

If a java.util.Hashtable object is found either inside the Subject or within the shared state area, verify that the following properties are present in the hashtable:

Property
com.ibm.wsspi.security.cred.uniqueId

Reference to the property
AttributeNameConstants.WSCREDENTIAL_UNIQUEID

Returns
java.util.String

Explanation

The value of the property must be a unique representation of the user. For the WebSphere Application Server default implementation, this property represents the information that is stored in the application authorization table. The information is located in the application deployment descriptor after it is deployed and user-to-role mapping is performed. See the expected format examples if the user to role mapping is performed using a lookup to a WebSphere Application Server user registry implementation.

If a third-party authorization provider overrides the user-to-role mapping, then the third-party authorization provider defines the format. To ensure compatibility with the WebSphere Application Server default implementation for the unique ID value, call the WebSphere Application Server public String getUniqueUserId(String userSecurityName) UserRegistry method.

Expected format examples

Table 42. Format examples.

This table gives some format examples when configuring inbound identity mapping.

Realm	Format (uniqueUserId)
Lightweight Directory Access Protocol (LDAP)	`ldaphost.austin.ibm.com:389/cn=user,o=ibm,c=us`
Windows	`MYWINHOST/S-1-5-21-963918322-163748893-4247568029-500`
UNIX	`MYUNIXHOST/32`

The com.ibm.wsspi.security.cred.uniqueId property is required.

Property

com.ibm.wsspi.security.cred.securityName

Reference to the property

AttributeNameConstants. WSCREDENTIAL_ SECURITYNAME

Returns

java.util.String

Explanation

This property searches for the securityName of the authentication user. This name is commonly called the *display name* or *short name*. WebSphere Application Server uses the securityName attribute for the getRemoteUser, getUserPrincipal and getCallerPrincipal application programming interfaces (APIs). To ensure compatibility with the WebSphere Application Server default implementation for the securityName value, call the WebSphere Application Server public String getUserSecurityName(String uniqueUserId) UserRegistry method.

Expected format examples

Table 43. Format examples. This table gives expected format examples.

Realm	Format (uniqueUserId)
LDAP	user (*LDAP UID*)
Windows	user (*Windows username*)
UNIX	user (*UNIX username*)

The com.ibm.wsspi.security.cred.securityName property is required.

Property

com.ibm.wsspi.security.cred.groups

Reference to the property

AttributeNameConstants. WSCREDENTIAL_GROUPS

Returns

java.util.ArrayList

Explanation

This key searches for the array list of groups to which the user belongs. The groups are specified in the *realm_name*/*user_name* format. The format of these groups is important as the groups are used by the WebSphere Application Server authorization engine for group-to-role mappings in the deployment descriptor. The format that is provided must match the format expected by the WebSphere Application Server default implementation. When you use a third-party authorization provider, you must use the format that is expected by the third-party provider. To ensure compatibility with the WebSphere Application Server default implementation for the unique group IDs value, call the WebSphere Application Server public List getUniqueGroupIds(String uniqueUserId) UserRegistry method.

Expected format examples for each group in the array list

Table 44. Format examples. This table gives expected format examples for each group in the array list.

Realm	Format
LDAP	`ldap1.austin.ibm.com:389/cn=group1,o=ibm,c=us`
Windows	`MYWINREALM/S-1-5-32-544`
UNIX	`MY/S-1-5-32-544`

The com.ibm.wsspi.security.cred.groups property is not required. A user is not required to have associated groups.

Property

com.ibm.wsspi.security.cred.cacheKey

Reference to the property

AttributeNameConstants. WSCREDENTIAL_CACHE_KEY

Returns

java.lang.Object

Explanation

This key property can specify an object that represents the unique properties of the login, including the user-specific information and the user dynamic attributes that might affect uniqueness. For example, when the user logs in from location A, which might affect their access control, the cache key needs to include location A so that the Subject that is received is the correct Subject for the current location.

This com.ibm.wsspi.security.cred.cacheKey property is not required. When this property is not specified, the cache lookup is the value that is specified for WSCREDENTIAL_UNIQUEID. When this information is found in the java.util.Hashtable object, WebSphere Application Server creates a Subject similar to the Subject that goes through the normal login process at least for LTPA. The new Subject contains a WSCredential object and a WSPrincipal object that is fully populated with the information found in the Hashtable object.

4. Add your custom login module into the RMI_INBOUND, WEB_INBOUND, and DEFAULT Java Authentication and Authorization Service (JAAS) system login configurations. Configure the RMI_INBOUND login configuration so that WebSphere Application Server loads your new custom login module first.
   a. Click **Security > Global security > Java Authentication and Authorization Service > System logins > RMI_INBOUND**
   b. Under Additional Properties, click **JAAS login modules > New** to add your login module to the RMI_INBOUND configuration.
   c. Return to the JAAS login modules panel for RMI_INBOUND.
   d. Click **Set order** to change the order that the login modules are loaded so that WebSphere Application Server loads your custom login module first. Use the **Move Up** or **Move Down** buttons to arrange the order of the login modules.

e. Repeat the previous three steps for the WEB_INBOUND and DEFAULT login configurations.

Results

This process configures identity mapping for an inbound request.

Example

The “Example: Custom login module for inbound mapping” topic shows a custom login module that creates a java.util.Hashtable hashtable that is based on the specified NameCallback callback. The java.util.Hashtable hashtable is added to the sharedState java.util.Map map so that the WebSphere Application Server login modules can locate the information in the hashtable.

Example: Custom login module for inbound mapping

This sample shows a custom login module that creates a java.util.Hashtable hashtable that is based on the specified NameCallback callback. The java.util.Hashtable hashtable is added to the sharedState java.util.Map map so that the WebSphere Application Server login modules can locate the information in the Hashtable.

```
public customLoginModule()
{

public void initialize(Subject subject, CallbackHandler callbackHandler,
   Map sharedState, Map options)
{
 // (For more information on initialization, see
 //  "Developing custom login modules for a system login configuration for JAAS" on page 442.)
 _sharedState = sharedState;
}

public boolean login() throws LoginException
{
 // (For more information on what to do during login, see
 //  "Developing custom login modules for a system login configuration for JAAS" on page 442.)

   // Handles the WSTokenHolderCallback to see if this is an initial or
   // propagation login.
 javax.security.auth.callback.Callback callbacks[] =
     new javax.security.auth.callback.Callback[3];
 callbacks[0] = new javax.security.auth.callback.NameCallback("");
 callbacks[1] = new javax.security.auth.callback.PasswordCallback(
     "Password: ", false);
 callbacks[2] = new com.ibm.websphere.security.auth.callback.
     WSCredTokenCallbackImpl("");
 callbacks[3] = new com.ibm.wsspi.security.auth.callback.
     WSTokenHolderCallback("");

 try
 {
  callbackHandler.handle(callbacks);
 }
 catch (Exception e)
 {
  // Handles the exception
 }

 // Determines which callbacks contain information
 boolean identitySwitched = false;
 String uid = ((NameCallback) callbacks[0]).getName();
 char password[] = ((PasswordCallback) callbacks[1]).getPassword();
 byte[] credToken = ((WSCredTokenCallbackImpl) callbacks[2]).getCredToken();
 java.util.List authzTokenList = ((WSTokenHolderCallback) callbacks[3]).
     getTokenHolderList();

 if (credToken != null)
 {
  try
  {
   String uniqueID = WSSecurityPropagationHelper.validateLTPAToken(credToken);
   String realm = WSSecurityPropagationHelper.getRealmFromUniqueID (uniqueID);
       // Set the string to the UID so you can use the information to either
       // map or login.
   uid = WSSecurityPropagationHelper.getUserFromUniqueID (uniqueid);
  }
  catch (Exception e)
  {
   // handle exception
  }
 }
 else if (uid == null)
```

```
{
 // The authentication data is not valid. You must have either UID
   // or CredToken
 throw new WSLoginFailedException("invalid authentication data.");
}
else if (uid != null && password != null)
{
    // This is a typical authentication. You can choose to map this ID to
    // another ID or you can skip it and allow WebSphere Application Server
    // to log in for you. When passwords are presented, be very careful not
    // to validate the password because this is the initial authentication.

 return true;
}

 // You can map this uid to something else and set the identitySwitched
 // boolean. If the identity is changed, clear the following propagated
 // attributes so they are not used incorrectly.
uid = myCustomMappingRoutine (uid);

// Clear the propagated attributes because they no longer apply to the new identity
if (identitySwitched)
{
 ((WSTokenHolderCallback) callbacks[3]).setTokenHolderList(null);
}
boolean requiresLogin = ((com.ibm.wsspi.security.auth.callback.
    WSTokenHolderCallback) callbacks[2]).getRequiresLogin();

if (requiresLogin || identitySwitched)
{
 // Retrieves the default InitialContext for this server.
 javax.naming.InitialContext ctx = new javax.naming.InitialContext();

 // Retrieves the local UserRegistry object.
 com.ibm.websphere.security.UserRegistry reg =
        (com.ibm.websphere.security.UserRegistry) ctx.lookup("UserRegistry");

 // Retrieves the registry uniqueID based on the uid that is specified
    // in the NameCallback.
 String uniqueid = reg.getUniqueUserId(uid);
  uid = WSSecurityPropagationHelper.getUserFromUniqueID (uniqueid);

 // Retrieves the display name from the user registry based on the uniqueID.
 String securityName = reg.getUserSecurityName(uid);

 // Retrieves the groups associated with this uniqueID.
 java.util.List groupList = reg.getUniqueGroupIds(uid);

 // Creates the java.util.Hashtable with the information that you gathered
    // from the UserRegistry.
 java.util.Hashtable hashtable = new java.util.Hashtable();
 hashtable.put(com.ibm.wsspi.security.token.AttributeNameConstants.
      WSCREDENTIAL_UNIQUEID, uniqueid);
    hashtable.put(com.ibm.wsspi.security.token.AttributeNameConstants.
      WSCREDENTIAL_SECURITYNAME, securityName);
 hashtable.put(com.ibm.wsspi.security.token.AttributeNameConstants.
      WSCREDENTIAL_GROUPS, groupList);

 // Adds a cache key that is used as part of the lookup mechanism for
 // the created Subject. The cache key can be an object, but has
 // an implemented toString method. Make sure the cacheKey contains enough
 // information to scope it to the user and any additional attributes you are
    // using. If you do not specify this property, the Subject is scoped to the
 // WSCREDENTIAL_UNIQUEID returned, by default.
 hashtable.put(com.ibm.wsspi.security.token.AttributeNameConstants.
      WSCREDENTIAL_CACHE_KEY, "myCustomAttribute" + uniqueid);
 // Adds the hashtable to the shared state of the Subject.
 _sharedState.put(com.ibm.wsspi.security.token.AttributeNameConstants.
      WSCREDENTIAL_PROPERTIES_KEY, hashtable);
}
else if (requiresLogin == false)
{
 // For more information on this section, see
    // "Security attribute propagation" on page 468.
 // If you added a custom Token implementation, you can search through the
    // token holder list for it to deserialize.
 // Note: Any Java objects are automatically deserialized by
    // wsMapDefaultInboundLoginModule

 for (int i=0; i<authzTokenList.size(); i++)
 {
  TokenHolder tokenHolder = (TokenHolder) authzTokenList.get(i);
  if (tokenHolder.getName().equals("com.acme.MyCustomTokenImpl"))
  {
     byte[] myTokenBytes = tokenHolder.getBytes();

         // Passes these bytes into the constructor of your implementation
         // class for deserialization.
     com.acme.MyCustomTokenImpl myTokenImpl = new com.acme.MyCustomTokenImpl(myTokenBytes);
  }
```

```
  }
 }
}

public boolean commit() throws LoginException
{
 // (For more information on what to do during a commit, see
  //   "Developing custom login modules for a system login configuration for JAAS" on page 442.)
}

// Defines your login module variables
com.ibm.wsspi.security.token.AuthorizationToken customAuthzToken = null;
com.ibm.wsspi.security.token.AuthenticationToken defaultAuthToken = null;
java.util.Map _sharedState = null;
}
```

Configuring outbound identity mapping to a different target realm

By default, when WebSphere Application Server makes an outbound request from one server to another server in a different security realm, the request is rejected. This topic details alternatives for enabling one server to send outbound requests to a target server in a different realm.

About this task

This outbound request is rejected to protect against a rogue server reading potentially sensitive information if successfully impersonating the home of the object. Select one of the following alternative procedures so that one server can send outbound requests to a target server in a different realm. When you are finished with a procedure on the administrative console, click **Apply**.

Procedure

- Do not perform mapping. Instead, allow the existing security information to flow to a trusted target server, even if the target server resides in a different realm. Complete the following steps in the administrative console:
  1. Click **Security > Global security**.
  2. Under RMI/IIOP security, click **CSIv2 outbound authentication**.
  3. Specify the target realms in the **Trusted target realms** field. You can specify each trusted target realm that is separated by a pipe (|) character. For example, specify *server_name*.*domain*:*port_number* for a Lightweight Directory Access Protocol (LDAP) server or the machine name for local operating system. If you want to propagate security attributes to a different target realm, you must specify that target realm in the **Trusted target realms** field.
- Use the Java Authentication and Authorization Service (JAAS) WSLogin application login configuration to create a basic authentication Subject that contains the credentials of the new target realm. This configuration enables you to log in with a realm, user ID, and password that are specific to the user registry of the target realm. You can provide the login information from within the Java Platform, Enterprise Edition (Java EE) application that is making the outbound request or from within the RMI_OUTBOUND system login configuration. These two login options are described in the following information:
  1. Use the WSLogin application login configuration from within the Java EE application to log in and get a Subject that contains the user ID and the password of the target realm. The application can wrap the remote call with a WSSubject.doAs call. For an example, see “Example: Using the WSLogin configuration to create a basic authentication subject” on page 465.
  2. Use the code sample in “Example: Using the WSLogin configuration to create a basic authentication subject” on page 465 from this plug point within the RMI_OUTBOUND login configuration. Every outbound Remote Method Invocation (RMI) request passes through this login configuration when it is enabled. Complete the following steps to enable and plug in this login configuration:
     a. Click **Security > Global security**.
     b. Under RMI/IIOP security, click **CSIv2 outbound authentication**.
     c. Select the **Custom outbound mapping** option. If the **Security Attribute Propagation** option is selected, then WebSphere Application Server is already using this login configuration and you do not need to enable custom outbound mapping.

d. Write a custom login module. For more information, see “Developing custom login modules for a system login configuration for JAAS” on page 442.

   The “Example: Sample login configuration for RMI_OUTBOUND” on page 466 shows a custom login module that determines whether the realm names match. In this example, the realm names do not match so the WSLoginmodule is used to create a basic authentication Subject based on custom mapping rules. The custom mapping rules are specific to the customer environment and must be implemented using a realm to user ID and password mapping utility.

e. Configure the RMI_OUTBOUND login configuration so that your new custom login module is first in the list.
   1) Click **Security > Global security**.
   2) Under Java Authentication and Authorization Service, click **System logins > RMI_OUTBOUND**
   3) Under Additional Properties, click **JAAS login modules > New** to add your login module to the RMI_OUTBOUND configuration.
   4) Return to the JAAS login modules panel for RMI_OUTBOUND.
   5) Click **Set order** to change the order that the login modules are loaded so that your custom login is loaded first.

- Add the use_realm_callback and use_appcontext_callback options to the outbound mapping module for WSLogin. To add these options, complete the following steps:
  1. Click **Security > Global security**.
  2. Under Java Authentication and Authorization Service, click **Application logins > WSLogin**.
  3. Under Additional properties, click **JAAS login modules > com.ibm.ws.security.common.auth.module.WSLoginModuleImpl**.
  4. Under Additional properties, click **Custom Properties > New**.
  5. On the Custom properties panel, enter `use_realm_callback` in the **Name** field and `true` in the **Value** field.
  6. Click **OK**.
  7. Click **New** to enter the second custom property.
  8. On the Custom properties panel, enter `use_appcontext_callback` in the **Name** field and `true` in the **Value** field.

  The following changes are made to the `security.xml` file:

```
<entries xmi:id="JAASConfigurationEntry_2" alias="WSLogin">
 <loginModules xmi:id="JAASLoginModule_2"
  moduleClassName="com.ibm.ws.security.common.auth.module.proxy.WSLoginModuleProxy"
  authenticationStrategy="REQUIRED">
  <options xmi:id="Property_2" name="delegate"
   value="com.ibm.ws.security.common.auth.module.WSLoginModuleImpl"/>
  <options xmi:id="Property_3" name="use_realm_callback" value="true"/>
  <options xmi:id="Property_4" name="use_appcontext_callback" value="true"/>
 </loginModules>
</entries>
```

Example: Using the WSLogin configuration to create a basic authentication subject

This example shows how to use the WSLogin application login configuration from within a Java 2 Platform, Enterprise Edition (J2EE) application to log in and get a Subject that contains the user ID and the password of the target realm.

```
javax.security.auth.Subject subject = null;

try
{
  // Create a login context using the WSLogin login configuration and specify a
  // user ID, target realm, and password. Note: If the target_realm_name is the
  // same as the current realm, an authenticated Subject is created. However, if
  // the target_realm_name is different from the current realm, a basic
  // authentication Subject is created that is not validated. This unvalidated
  // Subject is created so that you can send a request to the different target
  // realm with valid security credentials for that realm.
  javax.security.auth.login.LoginContext ctx = new LoginContext("WSLogin",
```

```
   new WSCallbackHandlerImpl("userid", "target_realm_name", "password"));

  // Note: The following code is an alternative that validates the user ID and
  // password specified against the target realm. The code performs a remote call
  // to the target server and will return  true if the user ID and password are
  // valid and false if the user ID and password are not valid. If false is
  // returned, a WSLoginFailedException exception is created. You can catch
  // that exception and perform a retry or stop the request from flowing by
  // allowing that exception to surface out of this login.

  // ALTERNATIVE LOGIN CONTEXT THAT VALIDATES THE USER ID AND PASSWORD TO THE
  // TARGET REALM

  /****  currently remarked out ****
  java.util.Map appContext = new java.util.HashMap();
              appContext.put(javax.naming.Context.INITIAL_CONTEXT_FACTORY,
                             "com.ibm.websphere.naming.WsnInitialContextFactory");
              appContext.put(javax.naming.Context.PROVIDER_URL,
                             "corbaloc:iiop:target_host:2809");

  javax.security.auth.login.LoginContext ctx = new LoginContext("WSLogin",
   new WSCallbackHandlerImpl("userid", "target_realm_name", "password", appContext));
  **** currently remarked out  ****/

  // Starts the login
  ctx.login();

  // Gets the Subject from the context
  subject = ctx.getSubject();
 }
 catch (javax.security.auth.login.LoginException e)
 {
  throw new com.ibm.websphere.security.auth.WSLoginFailedException (e.getMessage(), e);
 }

 if (subject != null)
 {
  // Defines a privileged action that encapsulates your remote request.
java.security.PrivilegedAction myAction = java.security.PrivilegedAction()
  {
   public Object run()
   {
    // Assumes a proxy is already defined. This example method returns a String
    return proxy.remoteRequest();
   }
  });

  // Starts this action using the basic authentication Subject needed for
     // the target realm security requirements.
  String myResult = (String) com.ibm.websphere.security.auth.WSSubject.doAs
        (subject, myAction);
 }
```

Example: Sample login configuration for RMI_OUTBOUND

This example shows a sample login configuration for RMI_OUTBOUND that determines whether the realm names match between two servers.

```
public customLoginModule()
{
 public void initialize(Subject subject, CallbackHandler callbackHandler,
     Map sharedState, Map options)
 {
     // (For more information on what to do during initialization, see
     // "Developing custom login modules for a system login configuration for JAAS" on page 442.)
 }

 public boolean login() throws LoginException
 {
     // (For more information on what to do during login, see
     // "Developing custom login modules for a system login configuration for JAAS" on page 442.)

  // Gets the WSProtocolPolicyCallback object
  Callback callbacks[] = new Callback[1];
   callbacks[0] = new com.ibm.wsspi.security.auth.callback.
          WSProtocolPolicyCallback("Protocol Policy Callback: ");

  try
  {
   callbackHandler.handle(callbacks);
  }
  catch (Exception e)
  {
   // Handles the exception
  }

     // Receives the RMI (CSIv2) policy object for checking the target realm
     // based upon information from the IOR.
     // Note: This object can be used to perform additional security checks.
```

```
   // See the application programming interface (API) documentation for
   // more information.
 csiv2PerformPolicy = (CSIv2PerformPolicy) ((WSProtocolPolicyCallback)callbacks[0]).
       getProtocolPolicy();

 // Checks if the realms do not match. If they do not match, then log in to
   // perform a mapping
 if (!csiv2PerformPolicy.getTargetSecurityName().equalsIgnoreCase(csiv2PerformPolicy.
        getCurrentSecurityName()))
 {
  try
  {
   // Do some custom realm -> user ID and password mapping
   MyBasicAuthDataObject myBasicAuthData = MyMappingLogin.lookup
         (csiv2PerformPolicy.getTargetSecurityName());

         // Creates the login context with basic authentication data gathered from
         // custom mapping
    javax.security.auth.login.LoginContext ctx = new LoginContext("WSLogin",
     new WSCallbackHandlerImpl(myBasicAuthData.userid,
       csiv2PerformPolicy.getTargetSecurityName(),
                 myBasicAuthData.password));

    // Starts the login
    ctx.login();

           // Gets the Subject from the context. This subject is used to replace
           // the passed-in Subject during the commit phase.
    basic_auth_subject = ctx.getSubject();
   }
   catch (javax.security.auth.login.LoginException e)
   {
    throw new com.ibm.websphere.security.auth.
           WSLoginFailedException (e.getMessage(), e);
   }
 }
}

public boolean commit() throws LoginException
{
    // (For more information on what to do during commit, see
    // "Developing custom login modules for a system login configuration for JAAS" on page 442.)

 if (basic_auth_subject != null)
 {
     // Removes everything from the current Subject and adds everything from the
     // basic_auth_subject
  try
  {
   public final Subject basic_auth_subject_priv = basic_auth_subject;
         // Do this in a doPrivileged code block so that application code
         // does not need to add additional permissions
   java.security.AccessController.doPrivileged(new java.security.
           PrivilegedExceptionAction()
   {
    public Object run() throws WSLoginFailedException
    {
             // Removes everything user-specific from the current outbound
             // Subject. This a temporary Subject for this specific invocation
             // so you are not affecting the Subject set on the thread. You may
             // keep any custom objects that you want to propagate in the Subject.
             // This example removes everything and adds just the new information
             // back in.
     try
     {
      subject.getPublicCredentials().clear();
      subject.getPrivateCredentials().clear();
      subject.getPrincipals().clear();
     }
     catch (Exception e)
     {
      throw new WSLoginFailedException (e.getMessage(), e);
     }

             // Adds everything from basic_auth_subject into the login subject.
             // This completes the mapping to the new user.
     try
     {
      subject.getPublicCredentials().addAll(basic_auth_subject.
                 getPublicCredentials());
      subject.getPrivateCredentials().addAll(basic_auth_subject.
                 getPrivateCredentials());
      subject.getPrincipals().addAll(basic_auth_subject.
                 getPrincipals());
     }
     catch (Exception e)
     {
      throw new WSLoginFailedException (e.getMessage(), e);
     }
```

```
      return null;
     }
    });
   }
   catch (PrivilegedActionException e)
   {
    throw new WSLoginFailedException (e.getException().getMessage(),
             e.getException());
   }
  }
 }

 // Defines your login module variables
 com.ibm.wsspi.security.csiv2.CSIv2PerformPolicy csiv2PerformPolicy = null;
 javax.security.auth.Subject basic_auth_subject = null;
}
```

Security attribute propagation

With *Security attribute propagation*, WebSphere Application Server can transport security attributes (authenticated Subject contents and security context information) from one server to another in your configuration. WebSphere Application Server might obtain these security attributes from either an enterprise user registry, which queries static attributes, or a custom login module, which can query static or dynamic attributes. Dynamic security attributes, which are custom in nature, might include the authentication strength that is used for the connection, the identity of the original caller, the location of the original caller, the IP address of the original caller, and so on.

Security attribute propagation provides propagation services using Java serialization for any objects that are contained in the Subject. However, Java code must be able to serialize and deserialize these objects. The Java programming language specifies the rules for how Java code can serialize an object. Because problems can occur when dealing with different platforms and versions of software, WebSphere Application Server also offers a token framework that enables custom serialization functionality. The token framework has other benefits that include the ability to identify the uniqueness of the token. This uniqueness determines how the Subject gets cached and the purpose of the token. The token framework defines four marker token interfaces that enable the WebSphere Application Server runtime to determine how to propagate the token.

Important: Any custom tokens that are used in this framework are not used by WebSphere Application Server for authorization or authentication. The framework serves as a way to notify WebSphere Application Server that you want these tokens propagated in a particular way. WebSphere Application Server handles the propagation details, but does not handle serialization or deserialization of custom tokens. Serialization and deserialization of these custom tokens are carried out by the implementation and handled by a custom login module.

With WebSphere Application Server Version 6.0 and later, a custom Java Authorization Contract for Container (JACC) provider can be configured to enforce access control for Java Platform, Enterprise Edition (Java EE) applications. A custom JACC provider can explore the custom security attributes in the caller JAAS subject in making access control decisions.

When a request is being authenticated, a determination is made by the login modules whether this request is an *initial login* or a *propagation login*. An initial login is the process of authenticating the user information, typically a user ID and password, and then calling the application programming interfaces (APIs) for the remote user registry to look up secure attributes that represent the user access rights. A propagation login is the process of validating the user information, typically a Lightweight Third Party Authentication (LTPA) token, and then deserializing a series of tokens that constitute both custom objects and token framework objects known to WebSphere Application Server.

The following marker tokens are introduced in the framework:

Authorization token
The authorization token contains most of the authorization-related security attributes that are propagated. The default authorization token is used by the WebSphere Application Server

authorization engine to make Java Platform, Enterprise Edition (Java EE) authorization decisions. Service providers can use custom authorization token implementations to isolate their data in a different token, perform custom serialization and de-serialization, and make custom authorization decisions using the information in their token at the appropriate time. For information on how to use and implement this token type, see “Using the default propagation token to propagate security attributes” on page 479 and “Implementing a custom propagation token for security attribute propagation” on page 888.

Single sign-on (SSO) token

A custom SingleSignonToken token that is added to the Subject is automatically added to the response as an HTTP cookie and contains the attributes sent back to web browsers. The token interface getName method with the getVersion method defines the cookie name. WebSphere Application Server defines a default SingleSignonToken token with the LtpaToken name and Version 2. The cookie name added is LtpaToken2. Do not add sensitive information, confidential information, or unencrypted data to the response cookie.

It is also recommended that any time that you use cookies, use the Secure Sockets Layer (SSL) protocol to protect the request. Using an SSO token, web users can authenticate once when accessing web resources across multiple WebSphere Application Servers. A custom SSO token extends this functionality by adding custom processing to the single sign-on scenario. For more information on SSO tokens, see “Implementing single sign-on to minimize web user authentications” on page 373. For information on how to use and implement this token type, see “Using the default single sign-on token with default or custom token factory to propagate security attributes” on page 484 and “Implementing a custom single sign-on token for security attribute propagation” on page 901.

Propagation token

The propagation token is not associated with the authenticated user so it is not stored in the Subject. Instead, the propagation token is stored on the thread and follows the invocation wherever it goes. When a request is sent outbound to another server, the propagation tokens on that thread are sent with the request and the tokens are run by the target server. The attributes that are stored on the thread are propagated regardless of the Java Platform, Enterprise Edition (Java EE) RunAs user switches.

The default propagation token monitors and logs all user switches and host switches. You can add additional information to the default propagation token using the WSSecurityHelper application programming interfaces (APIs). To retrieve and set custom implementations of a propagation token, you can use the WSSecurityPropagationHelper class. For information on how to use and implement this token type, see “Using the default propagation token to propagate security attributes” on page 479 and “Implementing a custom propagation token for security attribute propagation” on page 888.

Authentication token

The authentication token flows to downstream servers and contains the identity of the user. This token type serves the same function as the Lightweight Third Party Authentication (LTPA) token in previous versions. Although this token type is typically reserved for internal WebSphere Application Server purposes, you can add this token to the Subject and the token is propagated using the getBytes method of the token interface.

A custom authentication token is used solely for the purpose of the service provider that adds it to the Subject. WebSphere Application Server does not use it for authentication purposes because a default authentication token exists that is used for WebSphere Application Server authentication. This token type is available for the service provider to identify how the custom data uses the token to perform custom authentication decisions. For information on how to use and implement this token type, see “Default authentication token” on page 472 and “Implementing a custom authentication token for security attribute propagation” on page 909.

Kerberos authentication token

The Kerberos authentication token contains Kerberos credentials such as the Kerberos principal

name, GSSCredential and Kerberos delegation credential. This token is propagated to the downstream server. Although this token type is typically reserved for internal WebSphere Application Server purposes, if it contains the GSSCredential you can use the getGSSCredential method to extract the GSSCredential. You can then place it in the subject and it can be used for your application. This token is created when you authenticate to WebSphere Application Server with either SPNEGO web or Kerberos authentication.

Horizontal propagation versus downstream propagation

In WebSphere Application Server, both horizontal propagation, which uses single sign-on for web requests, and downstream propagation, which uses Remote Method Invocation over the Internet Inter-ORB Protocol (RMI/IIOP) to access enterprise beans, are available.

Horizontal propagation

In horizontal propagation, security attributes are propagated among front-end servers. The serialized security attributes, which are the Subject contents and the propagation tokens, can contain both static and dynamic attributes. The single sign-on (SSO) token stores additional system-specific information that is needed for horizontal propagation. The information contained in the SSO token tells the receiving server where the originating server is located and how to communicate with that server. Additionally, the SSO token also contains the key to look up the serialized attributes. To enable horizontal propagation, you must configure the single sign-on token and the web inbound security attribute propagation features. You can configure both of these features using the administrative console.

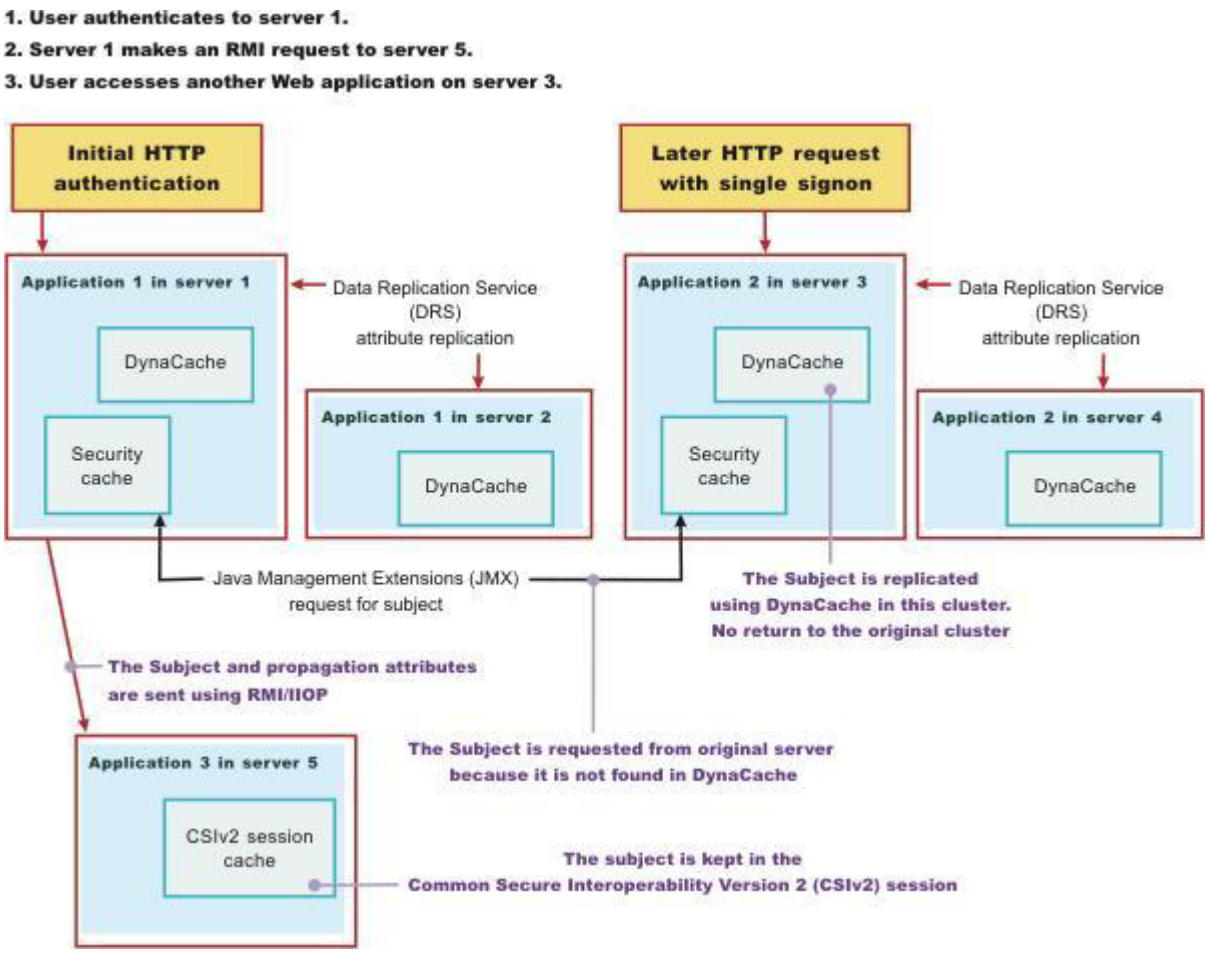


Figure 22. Horizontal propagation

Performance implications for horizontal propagation

The performance implications of the JMX remote call depends upon your environment. The JMX remote call is used for obtaining the original login attributes. Horizontal propagation reduces many of the remote user registry calls in cases where these calls cause the most performance problems for an application. However, the deserialization of these objects also might cause performance degradation, but this degradation might be less than the remote user registry calls. It is recommended that you test your environment with horizontal propagation enabled and disabled. In cases where you must use horizontal propagation for preserving original login attributes, test whether JMX provides better performance in your environment.

Downstream propagation

In *downstream propagation*, a Subject is generated at the web front-end server, either by a propagation login or a user registry login. WebSphere Application Server propagates the security information downstream for enterprise bean invocations when both Remote Method Invocation (RMI) outbound and inbound propagation are enabled.

Benefits of propagating security attributes

The security attribute propagation feature of WebSphere Application Server has the following benefits:

- Enables WebSphere Application Server to use the security attribute information for authentication and authorization purposes. The propagation of security attributes can eliminate the need for user registry

calls at each remote hop along an invocation. Previous versions of WebSphere Application Server propagated only the user name of the authenticated user, but ignored other security attribute information that needed to be regenerated downstream using remote user registry calls. To accentuate the benefits of this new functionality, consider the following example:

In previous releases, you might use a reverse proxy server (RPSS), such as WebSEAL, to authenticate the user, gather group information, and gather other security attributes. As stated previously, WebSphere Application Server accepted the identity of the authenticated user, but disregarded the additional security attribute information. To create a Java Authentication and Authorization Service (JAAS) Subject containing the needed WSCredential and WSPrincipal objects, WebSphere Application Server made 5 to 6 calls to the user registry. The WSCredential object contains various security information that is required to authorize a Java EE resource. The WSPrincipal object contains the realm name and the user that represents the principal for the Subject.

In the current release of the Application Server, information that is obtained from the reverse proxy server can be used by WebSphere Application Server and propagated downstream to other server resources without additional calls to the user registry. The retaining of the security attribute information enables you to protect server resources properly by making appropriate authorization and trust-based decisions User switches that occur because of Java EE RunAs configurations do not cause the application server to lose the original caller information. This information is stored in the PropagationToken located on the running thread.

- Enables third-party providers to plug in custom tokens. The token interface contains a getBytes method that enables the token implementation to define custom serialization, encryption methods, or both.
- Provides the ability to have multiple tokens of the same type within a Subject created by different providers. WebSphere Application Server can handle multiple tokens for the same purpose. For example, you might have multiple authorization tokens in the Subject and each token might have distinct authorization attributes that are generated by different providers.
- Provides the ability to have a unique ID for each token type that is used to formulate a more unique subject identifier than just the user name in cases where dynamic attributes might change the context of a user login. The token type has a getUniqueId() method that is used for returning a unique string for caching purposes. For example, you might need to propagate a location ID, which indicates the location from which the user logs into the system. This location ID can be generated during the original login using either an reverse proxy server or the WEB_INBOUND login configuration and added to the Subject prior to serialization. Other attributes might be added to the Subject as well and use a unique ID. All of the unique IDs must be considered for the uniqueness of the entire Subject. WebSphere Application Server has the ability to specify what is unique about the information in the Subject, which might affect how the user accesses the Subject later.

Default authentication token

Do not use the default authentication token in service provider code. This default token is used by the WebSphere Application Server run-time code only and is authentication mechanism specific.

Any modifications to this token by service provider code can potentially cause interoperability problems. If you need to create an authentication token for custom usage, see “Implementing a custom authentication token for security attribute propagation” on page 909 for more information.

Changing the token factory that is associated with the default authentication token

When WebSphere Application Server generates a default authentication token, the application server utilizes the TokenFactory class that is specified using the com.ibm.wsspi.security.token.authenticationTokenFactory property. To modify this property using the administrative console, complete the following steps:

1. Click **Security** > **Global security**.
2. Under Additional properties, click **Custom properties**.

The com.ibm.ws.security.ltpa.LTPATokenFactory token factory is the default for this property. The LTPATokenFactory token factory uses the DESede/ECB/PKCS5Padding cipher. This token factory creates an interoperable Lightweight Third Party Authentication (LTPA) token.

If you associate the com.ibm.ws.security.ltpa.LTPAToken2Factory token factory with the com.ibm.wsspi.security.token.authenticationTokenFactory property, the token is Advanced Encryption Standard (AES) encrypted. However, you need to weigh the performance against your security needs. You might add additional attributes to the authentication token in the Subject during a login that are available downstream.

If you need to perform your own signing and encryption of the default authentication token, you must implement the following classes:

- com.ibm.wsspi.security.ltpa.Token
- com.ibm.wsspi.security.ltpa.TokenFactory

Your token factory implementation instantiates (createToken) and validates (validateTokenBytes) your token implementation. You can use the LTPA keys that are passed into the initialize method of the token factory or you can use your own keys. If you use your own keys, they must be the same everywhere to validate the tokens that are generated using those keys. See the API documentation, available through a link on the front page of the information center, for more information on implementing your own custom token factory. To associate your token factory with the default authentication token using the administrative console, complete the following steps:

1. Click **Security** > **Global security**.
2. Under Additional properties, click **Custom properties**.
3. Locate the com.ibm.wsspi.security.token.authenticationTokenFactory property and verify that the value of this property matches your custom token factory implementation.
4. Verify that your implementation classes are put into the `install_dir/classes` directory so that the WebSphere Application Server class loader can load the classes.

Propagating security attributes among application servers

Use the security attribute propagation feature of WebSphere Application Server to send security attribute information regarding the original login to other servers using a token. This topic will help to configure WebSphere Application Server to propagate security attributes to other servers.

About this task

To fully enable security attribute propagation, you must configure the single sign-on (SSO), Common Secure Interoperability Version 2 (CSIv2) inbound, and CSIv2 outbound panels in the WebSphere Application Server administrative console. You can enable just the portions of security attribute propagation relevant to your configuration. For example, you can enable web propagation, which is propagation amongst front-end application servers, using either the push technique (DynaCache) or the pull technique (remote method to originating server).

You also can choose whether to enable Remote Method Invocation (RMI) outbound and inbound propagation, which is commonly called downstream propagation. Typically both types of propagation are enabled for any given cell. In some cases, you might want to choose a different option for a specific application server using the server security panel within the specific application server settings.

Restriction: To prevent propagating the same security attributes among application servers multiple times, WebSphere Application Server verifies that a Lightweight Third Party Authentication (LTPA) token does not exist. Two cases can occur. Absence of the LTPA token tells the Application Server that propagation can proceed. Presence of the LTPA token indicates that propagation has occurred if the LTPA token has been generated within the cluster. However,

in the second case, if the LTPA token is present, but has been generated by a server outside the cluster, such as by Tivoli Access Manager, Lotus Domino or a different Application Server cluster, security attributes are not propagated.

To access the server security panel in the administrative console, click **Servers > Application Servers > *server_name***. Under Security, click **Server security**.

Complete the following steps to configure WebSphere Application Server for security attribute propagation:

Procedure

1. Access the WebSphere Application Server administrative console by typing `http://server_name:port_number/ibm/console`. The administrative console address might differ if you have previously changed the port number.
2. Click **Security > Global security**.
3. Under Web security, click **Single sign-on (SSO)**.
4. Optional: Select the **Interoperability Mode** option if you need to interoperate with servers that do not support security attribute propagation. Servers that do not support security attribute propagation receive the Lightweight Third Party Authentication (LTPA) token and the Propagation token, but ignore the security attribute information that they do not understand.
5. Select the **Web inbound security attribute propagation** option. The Web inbound security attribute propagation option enables horizontal propagation, which allows the receiving SSO token to retrieve the login information from the original login server. If you do not enable this option, downstream propagation can occur if you enable the Security Attribute Propagation option on both the CSIv2 Inbound authentication and CSIv2 outbound authentication panels.

   Typically, you enable the web inbound security attribute propagation option if you need to gather dynamic security attributes set at the original login server that cannot be regenerated at the new front-end server. These attributes include any custom attributes that might be set in the PropagationToken token using the com.ibm.websphere.security.WSSecurityHelper application programming interfaces (APIs). You must determine whether enabling this option improves or degrades the performance of your system. While the option prevents some remote user registry calls, the deserialization and decryption of some tokens might impact performance. In some cases propagation is faster, especially if your user registry is the bottleneck of your topology. It is recommended that you measure the performance of your environment both using and not using this option. When you test the performance, it is recommended that you test in the operating environment of the typical production environment with the typical number of unique users accessing the system simultaneously.
6. Click **Security > Global security**. Under RMI/IIOP security, click **CSIv2 inbound authentication**. The Login configuration field specifies `RMI_INBOUND` as the system login configuration that is used for inbound requests. To add custom Java Authentication and Authorization Service (JAAS) login modules, complete the following steps:
   a. Click **Security > Global security**. Under Java Authentication and Authorization Service, click **System logins**. A list of the system login configurations is displayed. WebSphere Application Server provides the following pre-configured system login configurations: DEFAULT, LTPA, LTPA_WEB, RMI_INBOUND, RMI_OUTBOUND, SWAM, WEB_INBOUND, wssecurity.IDAssertion, and wssecurity.Signature. Do not delete these predefined configurations.

      Note: SWAM is deprecated in WebSphere Application Server Version 8.5 and will be removed in a future release.
   b. Click the name of the login configuration that you want to modify.
   c. Under Additional Properties, click **JAAS Login Modules**. The JAAS Login Modules panel is displayed, which lists all of the login modules that are processed in the login configuration. Do not

delete the required JAAS login modules. Instead, you can add custom login modules before or after the required login modules. If you add custom login modules, do not begin their names with com.ibm.ws.security.server.

You can specify the order in which the login modules are processed by clicking **Set Order**.

7. Select the **Security attribute propagation** option on the CSIv2 inbound authentication panel. When you select **Security Attribute Propagation**, the server advertises to other application servers that it can receive propagated security attributes from another server in the same realm over the Common Secure Interoperability version 2 (CSIv2) protocol.
8. Click **Security > Global security**. Under RMI/IIOP security, click **CSIv2 Outbound authentication**. The CSIv2 outbound authentication panel is displayed. The **Login configuration** field specifies `RMI_OUTBOUND` as the JAAS login configuration that is used for outbound configuration. You cannot change this login configuration. Instead, you can customize this login configuration by completing the substeps that are listed previously for CSIv2 Inbound authentication.
9. Optional: Verify that the **Security Attribute Propagation** option is selected if you want to enable outbound Subject and security context token propagation for the Remote Method Invocation (RMI) protocol. When you select this option, WebSphere Application Server serializes the Subject contents and the PropagationToken contents. After the contents are serialized, the server uses the CSIv2 protocol to send the Subject and PropagationToken token to the target servers that support security attribute propagation. If the receiving server does not support security attribute tokens, WebSphere Application Server sends the Lightweight Third Party Authentication (LTPA) token only.

   Important: WebSphere Application Server propagates only the objects within the Subject that it can serialize. The server propagates custom objects on a best-effort basis.

   When **Security Attribute Propagation** is enabled, WebSphere Application Server adds marker tokens to the Subject to enable the target server to add additional attributes during the inbound login. During the commit phase of the login, the marker tokens and the Subject are marked as read-only and cannot be modified thereafter.
10. Optional: Select the **Custom Outbound Mapping** option if you clear the **Security Attribute Propagation** option and you want to use the RMI_OUTBOUND login configuration. If neither the **Custom Outbound Mapping** option nor the **Security Attribute Propagation** option is selected, WebSphere Application Server does not call the RMI_OUTBOUND login configuration. If you need to plug in a credential mapping login module, you must select the **Custom Outbound Mapping** option.
11. Optional: Specify trusted target realm names in the **Trusted Target Realms** field. By specifying these realm names, information can be sent to servers that reside outside the realm of the sending server to support inbound mapping that is at these downstream servers. To perform outbound mapping to a realm different from the current realm, you must specify the realm in this field so that you can get to this point without having the request rejected because of a realm mismatch. If you need WebSphere Application Server to propagate security attributes to another realm when a request is sent, you must specify the realm name in the **Trusted Target Realms** field. Otherwise, the security attributes are not propagated to the unspecified realm. You can add multiple target realms by adding a pipe (|) delimiter between each entry.
12. Optional: Enable propagation for a pure client. For a pure client to propagate attributes added to the invocation Subject, you must add the following property to the `sas.client.props` file:

    ```
    com.ibm.CSI.rmiOutboundPropagationEnabled=true
    ```

    Note: The `sas.client.props` file is located at `<WAS-HOME>/profiles/<ProfileName>/properties>`.

Results

After completing these steps, you have configured WebSphere Application Server to propagate security attributes to other servers.

What to do next

If you need to disable security attribute propagation, determine whether you need to disable it for either the server level or the cell level.

Attention: Changes to the server-level settings override the cell settings.

To disable security attribute propagation on the server level, complete the following steps:

1. Click **Server > Application Servers > *server_name***.
2. Under Security, click **Server security**.
3. Select the **RMI/IIOP security for this server overrides cell settings** option.
4. Disable security attribute propagation for inbound requests by clicking **CSI inbound authentication** under Additional Properties and clearing the **Security attribute propagation** option.
5. Disable security attribute propagation for outbound requests by clicking **CSI outbound authentication** under Additional Properties and clearing the **Security attribute propagation** option.

To disable security attribute propagation on the cell level, undo each of the steps that you completed to enable security attribute propagation in this task.

Using the default authorization token to propagate security attributes

This topic explains how WebSphere Application Server uses the default authorization token. Consider using the default authorization token when you are looking for a place to add string attributes that get propagated downstream.

About this task

However, make sure that the attributes you add to the authorization token are specific to the user that is associated with the authenticated Subject. If they are not specific to a user, the attributes probably belong in the propagation token, which is also propagated with the request. For more information on the propagation token, see “Using the default propagation token to propagate security attributes” on page 479. To add attributes into the authorization token, you must plug in a custom login module into the various system login modules that are configured. Any login module configuration that has the com.ibm.ws.security.server.lm.wsMapDefaultInboundLoginModule implementation configured can receive propagated information and can generate propagation information that can be sent outbound to another server.

If propagated attributes are not presented to the login configuration during an initial login, a default authorization token is created in the wsMapDefaultInboundLoginModule login module after the login occurs in the ltpaLoginModule login module. You can obtain a reference to the default authorization token from the login method using the sharedState hashmap. You must plug in the custom login module after the wsMapDefaultInboundLoginModule implementation for WebSphere Application Server to see the default authorization token.

For more information on the Java Authentication and Authorization Service (JAAS) programming model, see the Security: Resources for learning article.

Procedure

- Obtain a reference to the default authorization token from the login method.
- Add attributes to the token.
- Read existing attributes used for authorization.
- Ensure that your custom login module code is trusted. Whenever you plug a custom login module into the WebSphere Application Server login infrastructure, you must ensure that the code is trusted. When you add the login module into the *app_server_root*/classes directory, it has Java 2 Security

AllPermissions permissions. It is recommended that you add your login module and other infrastructure classes into a private directory. However, if you use a private directory, modify the `$(WAS_INSTALL_ROOT)/properties/server.policy` file so that the private directory, Java archive (JAR) file, or both have the permissions that are needed to run the application programming interfaces (API) that are called from the login module. Because the login module might run after the application code on the call stack, you might consider adding a doPrivileged code block so that you do not need to add additional permissions to your applications.

- Modify the authorization token factory to use a token factory other than the default token factory.

  When WebSphere Application Server generates a default authorization token, the application server utilizes the TokenFactory class that is specified using the com.ibm.wsspi.security.token.authorizationTokenFactory property.

  The com.ibm.ws.security.ltpa.AuthzPropTokenFactory token factory is the default. This token factory encodes the data, but does not encrypt the data in the authorization token. Because the authorization token typically flows over Common Secure Interoperability Version 2 (CSIv2) using Secure Sockets Layer (SSL), encrypting the token is not necessary. However, if you need additional security for the authorization token, you can associate a different token factory implementation with this property to get encryption. For example, if you associate the com.ibm.ws.security.ltpa.LTPAToken2Factory token factory with this property, the token uses Advanced Encryption Standard (AES) encryption. However, you need to weigh the performance impacts against your security needs. Adding sensitive information to the authorization token is one reason to change the token factory implementation to something that encrypts rather than just encodes.

  1. Open the administrative console.
  2. Click **Security > Global security**.
  3. Under Additional properties, click **Custom properties**.

- Perform your own signing and encryption of the default authorization token.

  If you want to perform your own signing and encryption of the default authorization token, you must implement the following classes:

  – com.ibm.wsspi.security.ltpa.Token
  – com.ibm.wsspi.security.ltpa.TokenFactory

  Your token factory implementation instantiates and validates your token implementation. You can use the Lightweight Third Party Authentication (LTPA) keys that are passed into the initialize method of the token factory or you can use your own keys. If you use your own keys, they must be the same everywhere to validate the tokens that are generated using those keys. See the API documentation, that is available through a link on the front page of the information center, for more information on implementing your own custom token factory.

- Associate your token factory with the default authorization token.

  To associate your token factory with the default authorization token, using the administrative console, complete the following steps:

  1. Click **Security > Global security**.
  2. Under Additional properties, click **Custom properties**.
  3. Locate the com.ibm.wsspi.security.token.authorizationTokenFactory property and verify that the value of this property matches your custom token factory implementation.
  4. Verify that your implementation classes are put into the *app_server_root*`/classes` directory so that the WebSphere Application Server class loader can load the classes.
  5. Verify that your implementation classes are put into the `${USER_INSTALL_ROOT}/classes` directory so that the WebSphere Application Server class loader can load the classes.

Example

The following example shows the complete task of obtaining a reference to the default authorization token from the login method, adding attributes to the token, and reading from the existing attributes that are used for authorization.

```
public customLoginModule()
{
 public void initialize(Subject subject, CallbackHandler callbackHandler,
         Map sharedState, Map options)
 {
     // (For more information on initialization, see
     // "Developing custom login modules for a system login configuration for JAAS" on page 442.)

  // Get a reference to the sharedState map that is passed in during initialization.
  _sharedState = sharedState;
 }

 public boolean login() throws LoginException
 {
     // (For more information on what to do during login, see
     // "Developing custom login modules for a system login configuration for JAAS" on page 442.)

  // Look for the default AuthorizationToken in the shared state
  defaultAuthzToken  = (com.ibm.wsspi.security.token.AuthorizationToken)
       sharedState.get
     (com.ibm.wsspi.security.auth.callback.Constants.WSAUTHZTOKEN_KEY);

  // Might not always have one of these generated. It depends on the login
     // configuration setup.
  if (defaultAuthzToken != null)
  {
   try
   {
    // Add a custom attribute
    defaultAuthzToken.addAttribute("key1", "value1");

    // Determine all of the attributes and values that exist in the token.
    java.util.Enumeration listOfAttributes = defaultAuthorizationToken.
              getAttributeNames();

    while (listOfAttributes.hasMoreElements())
    {
     String key = (String) listOfAttributes.nextElement();

     String[] values = (String[]) defaultAuthorizationToken.getAttributes (key);

     for (int i=0;  i<values.length; i++)
     {
      System.out.println ("Key: " + key + ", Value[" + i + "]: "
                 + values[i]);
     }
    }

    // Read the existing uniqueID attribute.
    String[]  uniqueID = defaultAuthzToken.getAttributes
      (com.ibm.wsspi.security.token.AttributeNameConstants.
               WSCREDENTIAL_UNIQUEID);

     // Getthe uniqueID from the String[]
     String unique_id = (uniqueID != null &&
                uniqueID[0] != null) ? uniqueID[0] : "";

    // Read the existing expiration attribute.
    String[]  expiration = defaultAuthzToken.getAttributes
      (com.ibm.wsspi.security.token.AttributeNameConstants.
               WSCREDENTIAL_EXPIRATION);

     // An example of getting a long expiration value from the string array.
     long expire_time = 0;
     if (expiration != null && expiration[0] != null)
      expire_time = Long.parseLong(expiration[0]);

    // Read the existing display name attribute.
    String[]  securityName = defaultAuthzToken.getAttributes
      (com.ibm.wsspi.security.token.AttributeNameConstants.
               WSCREDENTIAL_SECURITYNAME);

     // Get the display name from the String[]
     String display_name = (securityName != null &&
                securityName[0] != null) ? securityName[0] : "";


    // Read the existing long securityName attribute.
    String[]  longSecurityName = defaultAuthzToken.getAttributes
     (com.ibm.wsspi.security.token.AttributeNameConstants.
             WSCREDENTIAL_LONGSECURITYNAME);
```

```
    // Get the long security name from the String[]
    String long_security_name = (longSecurityName != null &&
              longSecurityName[0] != null) ? longSecurityName[0] : "";


    // Read the existing group attribute.
    String[]  groupList = defaultAuthzToken.getAttributes
      (com.ibm.wsspi.security.token.AttributeNameConstants.
               WSCREDENTIAL_GROUPS);

    // Get the groups from the String[]
    ArrayList groups = new ArrayList();
    if (groupList != null)
    {
     for (int i=0; i<groupList.length; i++)
     {
      System.out.println ("group[" + i + "] = " + groupList[i]);
      groups.add(groupList[i]);
     }
    }
   }
   catch (Exception e)
   {
    throw new WSLoginFailedException (e.getMessage(), e);
   }
  }

 }

 public boolean commit() throws LoginException
 {
  // (For more information on what to do during commit, see
     // "Developing custom login modules for a system login configuration for JAAS" on page 442.)

 }

 private java.util.Map _sharedState = null;
 private com.ibm.wsspi.security.token.AuthorizationToken defaultAuthzToken = null;
}
```

.

Using the default propagation token to propagate security attributes

A default propagation token is located on the running thread for applications and the security infrastructure to use. The product propagates this default propagation token downstream and the token stays on the thread where the invocation lands at each hop.

About this task

The data is available from within the container of any resource where the propagation token lands. Remember that you must enable the propagation feature at each server where a request is sent for propagation to work. Make sure that you enable security attribute propagation for all of the cells in your environment where you want propagation

There is a WSSecurityHelper class that has application programming interfaces (APIs) for accessing the PropagationToken attributes. This topic documents the usage scenarios and includes examples. A close relationship exists between the propagation token and the work area feature. The main difference between these features is that after you add attributes to the propagation token, you cannot change the attributes. You cannot change these attributes so that the security runtime can add auditable information and have that information remain there for the life of the invocation. Any time that you add an attribute to a specific key, an ArrayList object is stored to hold that attribute. Any new attribute that is added with the same key is added to the ArrayList object. When you call getAttributes, the ArrayList object is converted to a String array and the order is preserved. The first element in the String array is the first attribute added for that specific key.

In the default propagation token, a change flag is kept that logs any data changes to the token. These changes are tracked to enable WebSphere Application Server to know when to send the authentication information downstream again so that the downstream server has those changes. Normally, Common Secure Interoperability Version 2 (CSIv2) maintains a session between servers for an authenticated client. If the propagation token changes, a new session is generated and subsequently a new authentication

occurs. Frequent changes to the propagation token during a method cause frequent downstream calls. If you change the token prior to making many downstream calls or you change the token between each downstream call, you might impact security performance.

Procedure

- Obtain the server list from the default propagation token.

  Every time the propagation token is propagated and used to create the authenticated Subject, either horizontally or downstream, the name of the receiving application server is logged into the propagation token. The format of the host is `"Cell:Node:Server"`, which provides you access to the cell name, node name, and server name of each application server that receives the invocation.

  The following code provides you with this list of names and can be called from a Java 2 Platform, Enterprise Edition (J2EE) application.

  The format of each server in the list is: *cell*:*node_name*:*server_name*. The output, for example, is: `myManager:node1:server1`

```
String[] server_list = null;

// If security is disabled on this application server, do not bother checking
if (com.ibm.websphere.security.WSSecurityHelper.isServerSecurityEnabled())
{
 try
 {
  // Gets the server_list string array
  server_list = com.ibm.websphere.security.WSSecurityHelper.getServerList();
 }
 catch (Exception e)
 {
  // Performs normal exception handling for your application
 }

 if (server_list != null)
 {
  // print out each server in the list, server_list[0] is the first server
  for (int i=0; i<server_list.length; i++)
  {
   System.out.println("Server[" + i + "] = " + server_list[i]);
  }
 }
}
```

- Obtain the list of callers, using the getCallerList API.

  A default propagation token is generated any time an authenticated user is set on the running thread or anyone tries to add attributes to the propagation token. Whenever an authenticated user is set on the thread, the user is logged in the default propagation token. At times, the same user might be logged in multiple times if the RunAs user is different from the caller. The following list provides the rules that are used to determine if a user that is added to the thread gets logged into the propagation token:

  – The current Subject must be authenticated. For example, an unauthenticated Subject is not logged.
  – The current authenticated Subject is logged if a Subject is not previously logged.
  – The current authenticated Subject is logged if the last authenticated Subject that is logged does not contain the same user.
  – The current authenticated Subject is logged on each unique application server that is involved in the propagation process.

  The following code sample shows how to use the getCallerList API.

The format of each caller in the list is: *cell*:*node_name*:*server_name*:*realm*:*port_number*/*securityName*. The output, for example, is: `myManager:node1:server1:ldap.austin.ibm.com:389/jsmith`.

```
String[] caller_list = null;

// If security is disabled on this application server, do not check the caller list
if (com.ibm.websphere.security.WSSecurityHelper.isServerSecurityEnabled())
{
 try
 {
  // Gets the caller_list string array
  caller_list = com.ibm.websphere.security.WSSecurityHelper.getCallerList();
 }
 catch (Exception e)
 {
  // Performs normal exception handling for your application
 }

 if (caller_list != null)
 {
  // Prints out each caller in the list, caller_list[0] is the first caller
  for (int i=0; i<caller_list.length;i++)
  {
   System.out.println("Caller[" + i + "] = " + caller_list[i]);
  }
 }
}
```

- Obtain the security name of the first authenticated user, using the getFirst Caller API.

  Whenever you want to know which authenticated caller started the request, you can call the getFirstCaller method and the caller list is parsed. However, this method returns the security name of the caller only. If you need to know more than the security name, call the getCallerList method and retrieve the first entry in the String array. This entry provides all the caller information.

  The following code sample retrieves the security name of the first authenticated caller using the getFirstCaller API.

  The output, for example, is: `jsmith`.

```
String first_caller = null;

// If security is disabled on this application server, do not bother checking
if (com.ibm.websphere.security.WSSecurityHelper.isServerSecurityEnabled())
{
 try
 {
  // Gets the first caller
  first_caller = com.ibm.websphere.security.WSSecurityHelper.getFirstCaller();

  // Prints out the caller name
  System.out.println("First caller: " + first_caller);
 }
 catch (Exception e)
 {
  // Performs normal exception handling for your application
 }
}
```

- Obtain the name of the first application server for a request, using the getFirstServer method.

Whenever you want to know what the first application server is for this request, call the getFirstServer method directly.

The following code sample retrieves the name of the first application server using the getFirstServer API.

The output, for example, is: `myManager:node1:server1`.

```
String first_server = null;

// If security is disabled on this application server, do not bother checking
if (com.ibm.websphere.security.WSSecurityHelper.isServerSecurityEnabled())
{
 try
 {
  // Gets the first server
  first_server = com.ibm.websphere.security.WSSecurityHelper.getFirstServer();

  // Prints out the server name
  System.out.println("First server: " + first_server);
 }
 catch (Exception e)
 {
  // Performs normal exception handling for your application
 }
}
```

- Add custom attributes to the default propagation token, using the addPropagationAttribute API.

  You can add custom attributes to the default propagation token for application usage. This token follows the request downstream so that the attributes are available when needed. When you use the default propagation token to add attributes, you must understand the following issues:

  – Adding information to the propagation token affects CSIv2 session caching. Add information sparingly between remote requests.
  – After you add information with a specific key, the information cannot be removed.
  – You can add as many values to a specific key as you need. However, all of the values must be available from a returned String array in the order that they were added.
  – The propagation token is available only on servers where propagation and security are enabled.
  – The Java 2 Security javax.security.auth.AuthPermission wssecurity.addPropagationAttribute attribute is needed to add attributes to the default propagation token.
  – An application cannot use keys that begin with either com.ibm.websphere.security or com.ibm.wsspi.security. These prefixes are reserved for system usage.

  The following code sample shows how to use the addPropagationAttribute API.

```
// If security is disabled on this application server,
    // do not check the status of server security
if (com.ibm.websphere.security.WSSecurityHelper.isServerSecurityEnabled())
{
 try
 {
  // Specifies the key and values
  String key = "mykey";
  String value1 = "value1";
  String value2 = "value2";

  // Sets key, value1
      com.ibm.websphere.security.WSSecurityHelper.
```

```
        addPropagationAttribute (key, value1);

    // Sets key, value2
    String[] previous_values = com.ibm.websphere.security.WSSecurityHelper.
           addPropagationAttribute (key, value2);

    // Note: previous_values should contain value1
   }
   catch (Exception e)
   {
    // Performs normal exception handling for your application
   }
  }
```

- Obtain your custom attributes with the get PropagationAttributes API.

  Custom attributes are added to the default propagation token using the addPropagationAttribute API. Retrieve these attributes using the getPropagationAttributes API. This token follows the request downstream so the attributes are available when needed. When you use the default propagation token to retrieve attributes, you must understand the following issues:

  – The propagation token is available only on servers where propagation and security are enabled.
  – The Java 2 Security javax.security.auth.AuthPermission "wssecurity.getPropagationAttributes" permission is needed to retrieve attributes from the default propagation token.

  See Adding custom attributes to the default PropagationToken to add attributes using the addPropagationAttributes API.

  The following code sample shows how to use the getPropagationAttributes API.

```
 // If security is disabled on this application server, do not bother checking
 if (com.ibm.websphere.security.WSSecurityHelper.isServerSecurityEnabled())
 {
  try
  {
   String key = "mykey";
   String[] values = null;

   // Sets key, value1
       values = com.ibm.websphere.security.WSSecurityHelper.
       getPropagationAttributes (key);

   // Prints the values
   for (int i=0; i<values.length; i++)
   {
    System.out.println("Value[" + i + "] = " + values[i]);
   }
  }
  catch (Exception e)
  {
   // Performs normal exception handling for your application
  }
 }
```

  The output, for example, is:

```
Value[0] = value1
Value[1] = value2
```

- Modify the propagation token factory configuration to use a token factory other than the default token factory.

When WebSphere Application Server generates a default propagation token, the Application Server utilizes the TokenFactory class that is specified using the com.ibm.wsspi.security.token.propagationTokenFactory property.

The default token factory that is specified for this property is called com.ibm.ws.security.ltpa.AuthzPropTokenFactory. This token factory encodes the data in the propagation token and does not encrypt the data. Because the propagation token typically flows over CSIv2 using Secure Sockets Layer (SSL), encrypting the token is not required. However, if you need additional security for the propagation token, you can associate a different token factory implementation with this property to get encryption. For example, if you choose to associate the com.ibm.ws.security.ltpa.LTPAToken2Factory token factory with this property, the token is AES encrypted. However, you need to weigh the performance impacts against your security needs. Adding sensitive information to the propagation token is a good reason to change the token factory implementation to something that encrypts rather than just encodes.

1. Open the administrative console.
2. Click **Security > Global security**.
3. Click **Custom properties**.

- Perform your own signing and encryption of the default propagation token.

  If you want to perform your own signing and encryption of the default propagation token, you must implement the following classes:

  – com.ibm.wsspi.security.ltpa.Token
  – com.ibm.wsspi.security.ltpa.TokenFactory

  Your token factory implementation instantiates and validates your token implementation. You can choose to use the Lightweight Third Party Authentication (LTPA) keys and have them pass into the initialize method of the token factory, or you can use your own keys. If you use your own keys, they must be the same everywhere to validate the tokens that are generated using those keys. See the API documentation, available through a link on the front page of the information center, for more information on implementing your own custom token factory.

- Associate your token factory with the default propagation token.
  1. Open the administrative console.
  2. Click **Security > Global security**.
  3. Click **Custom properties**.
  4. Locate the com.ibm.wsspi.security.token.propagationTokenFactory property and verify that the value of this property matches your custom token factory implementation.
  5. Verify that your implementation classes are put into the *app_server_root*/`classes` directory so that the WebSphere Application Server class loader can load the classes.
  6. Verify that your implementation classes are located in the `${USER_INSTALL_ROOT}/classes` directory so that the WebSphere Application Server class loader can load the classes.

Example

Using the default single sign-on token with default or custom token factory to propagate security attributes

Do not use the default single sign-on token in service provider code. This default token is used by the WebSphere Application Server run-time code only.

Before you begin

Size limitations exist for this token when it is added as an HTTP cookie. If you need to create an HTTP cookie using this token framework, you can implement a custom single sign-on token. To implement a custom single sign-on token see “Implementing a custom single sign-on token for security attribute propagation” on page 901 for more information.

Procedure

- Modify the single sign-on token factory configuration to use a token factory other than the default token factory.

  When the default single sign-on token is generated, the application server utilizes the TokenFactory class that is specified using the com.ibm.wsspi.security.token.singleSignonTokenFactory property. Use the administrative console to modify the property.

  The com.ibm.ws.security.ltpa.LTPAToken2Factory token factory is the default that is specified for this property. This token factory creates a single sign-on (SSO) token called LtpaToken2, which WebSphere Application Server uses for propagation. This token factory uses the AES/CBC/PKCS5Padding cipher.

  1. Open the administrative console.
  2. Click **Security > Global security**.
  3. Under Authentication, click **Custom properties**.
- Perform your own signing and encryption of the default single sign-on token.

  If you need to perform your own signing and encryption of the default single sign-on token, you must implement the following classes:

  – com.ibm.wsspi.security.ltpa.Token

  – com.ibm.wsspi.security.ltpa.TokenFactory

  Your token factory implementation instantiates (createToken) and validates (validateTokenBytes) your token implementation. You can use the Lightweight Third-Party Authentication (LTPA) keys passed into the initialize method of the token factory or you can use your own keys. If you use your own keys, they must be the same everywhere to validate the tokens that are generated using those keys. See the API reference information for more information on implementing your own custom token factory.
- Associate your own token factory with the default single sign-on token.
  1. Open the administrative console.
  2. Click **Security > Global security**.
  3. Under Authentication, click **Custom properties**.
  4. Locate the com.ibm.wsspi.security.token.singleSignonTokenFactory property and verify that the value of this property matches your custom TokenFactory implementation.
  5. Verify that your implementation classes are put into the *app_server_root*/`classes` directory so that the WebSphere Application Server class loader can load the classes.
  6. Verify that your implementation classes are located in the `${USER_INSTALL_ROOT}/classes` directory so that the WebSphere Application Server class loader can load the classes.

Configuring the authentication cache

The security authentication cache affects the frequency of rehashing and the distribution of the hash algorithms.

About this task

To configure the authentication cache properties, complete the following steps:

Procedure

1. Click **Servers > Application Servers > *server_name*** .
2. Under Server infrastructure, click **Java and Process Management > Process definition**.
3. Under Additional properties, click **Java Virtual Machine > Custom Properties**.
4. Click **New** to specify a new custom property.

What to do next

For information on the supported authentication cache properties, see “Authentication cache settings” on page 157.

Configuring Common Secure Interoperability Version 2 (CSIV2) inbound and outbound communication settings

WebSphere Application Server enables you to specify Internet Inter-ORB Protocol (IIOP) authentication for both inbound and outbound authentication requests. For inbound requests, you can specify the type of accepted authentication, such as basic authentication. For outbound requests, you can specify properties such as type of authentication, identity assertion or login configurations that are used for requests to downstream servers.

About this task

Complete the following steps to configure Common Secure Interoperability Version 2 (CSIV2) and Security Authentication Service (SAS).

Important: SAS is supported only between Version 6.0.x and previous version servers that have been federated in a Version 6.1 cell.

Procedure

1. Determine how to configure security inbound and outbound at each point in your infrastructure.

   For example, you might have a Java client communicating with an Enterprise JavaBeans (EJB) application server, which in turn communicates to a downstream EJB application server.

   The Java client utilizes the `sas.client.props` file to configure outbound security. Pure clients must configure outbound security only.

   The upstream EJB application server configures inbound security to handle the correct type of authentication from the Java client. The upstream EJB application server utilizes the outbound security configuration when going to the downstream EJB application server.

   This type of authentication might be different than what you expect from the Java client into the upstream EJB application server. Security might be tighter between the pure client and the first EJB server, depending on your infrastructure. The downstream EJB server utilizes the inbound security configuration to accept requests from the upstream EJB server. These two servers require similar configuration options as well. If the downstream EJB application server communicates to other downstream servers, the outbound security might require a special configuration.
2. Specify the type of authentication.

   By default, authentication by a user ID and password is performed.

   Both Java client certificate authentication and identity assertion are disabled by default. If you want this type of authentication performed at every tier, use the CSIv2 authentication protocol configuration as is. However, if you have any special requirements where some servers authenticate differently from other servers, consider how to configure CSIv2 to its best advantage.
3. Configure clients and servers.

   Configuring a pure Java client is done through the `sas.client.props` file, where properties are modified.

   Configuring servers is always done from the administrative console or scripting, either from the security navigation for cell-level configurations or from the server security of the application server for server-level configurations. If you want some servers to authenticate differently from others, modify some of the server-level configurations. When you modify the server-level configurations, you are overriding the cell-level configurations.

What to do next

Use CSIV2 inbound communications settings for configuring the type of authentication information that is contained in an incoming request or transport.

Use CSIV2 outbound communications settings to specify the features that a server supports when acting as a client to another downstream server.

Configuring Common Secure Interoperability Version 2 inbound communications

Inbound communications refers to the configuration that determines the type of accepted authentication for inbound requests. This authentication is advertised in the interoperable object reference (IOR) that the client retrieves from the name server.

Procedure

1. Start the administrative console.
2. Click **Security > Global security**.
3. Under RMI/IIOP security, click **CSIv2 inbound communications**.
4. Consider the following layers of security:
   - Identity assertion (attribute layer).

     When selected, this server accepts identity tokens from upstream servers. If the server receives an identity token, the identity is taken from an originating client. For example, the identity is in the same form that the originating client presented to the first server. An upstream server sends the identity of the originating client. The format of the identity can be either a principal name, a distinguished name, or a certificate chain. In some cases, the identity is anonymous. It is important to trust the upstream server that sends the identity token because the identity authenticates on this server. Trust of the upstream server is established either using Secure Sockets Layer (SSL) client certificate authentication or basic authentication. You must select one of the two layers of authentication in both inbound and outbound authentication when you choose identity assertion.

     The server ID is sent in the client authentication token with the identity token. The server ID is checked against the trusted server ID list. If the server ID is on the trusted server list, the server ID is authenticated. If the server ID is valid, the identity token is put into a credential and used for authorization of the request.

     Note: When identity assertion is enabled, message layer or transport layer should be enabled also. For server-to-server communication, besides enabling transport layer/client authentication, identity assertion or message layer should be enabled also.

     For more information, refer to Identity assertion.
   - Message layer:

     Basic authentication (GSSUP):

     This type of authentication is the most typical. The user ID and password or authenticated token is sent from a pure client or from an upstream server. When a user ID and password are received at the server, they are authenticated with the user registry of the downstream server.

     Lightweight Third Party Authentication (LTPA):

     In this case, an LTPA token is sent from the upstream server. Note that if you choose LTPA, then both servers must share the same LTPA keys

     Kerberos (KRB5):

     To select Kerberos, the active authentication mechanism must be Kerberos. In this case, a Kerberos token is sent from the upstream server.

     For more information, read about Message layer authentication.
   - Secure Sockets Layer client certificate authentication (transport layer).

The SSL client certificate is used to authenticate instead of using user ID and Password. If a server delegates an identity to a downstream server, the identity comes from either the message layer (a client authentication token) or the attribute layer (an identity token), and not from the transport layer through the client certificate authentication.

A client has an SSL client certificate that is stored in the keystore file of the client configuration. When SSL client authentication is enabled on this server, the server requests that the client send the SSL client certificate when the connection is established. The certificate chain is available on the socket whenever a request is sent to the server. The server request interceptor gets the certificate chain from the socket and maps this certificate chain to a user in the user registry. This type of authentication is optimal for communicating directly from a client to a server. However, when you have to go downstream, the identity typically flows over the message layer or through identity assertion.

5. Consider the following points when deciding what type of authentication to accept:
   - A server can receive multiple layers simultaneously, so an order of precedence rule decides which identity to use. The identity assertion layer has the highest priority, the message layer follows, and the transport layer has the lowest priority. The SSL client certificate authentication is used when it is the only layer provided. If the message layer and the transport layer are provided, the message layer is used to establish the identity for authorization. The identity assertion layer is used to establish precedence when provided.
   - Does this server usually receive requests from a client, from a server, or both? If the server always receives requests from a client, identity assertion is not needed. You can choose either the message layer, the transport layer, or both. You also can decide when authentication is required or just supported. To select a layer as required, the sending client must supply this layer, or the request is rejected. However, if the layer is only supported, the layer might not be supplied.
   - What kind of client identity is supplied? If the client identity is client certificates authentication and you want the certificate chain to flow downstream so that it maps to the downstream server user registries, identity assertion is the appropriate choice. Identity assertion preserves the format of the originating client. If the originating client authenticated with a user ID and password, a principal identity is sent. If authentication is done with a certificate, the certificate chain is sent.

     In some cases, if the client authenticated with a token and a Lightweight Directory Access Protocol (LDAP) server is the user registry, then a distinguished name (DN) is sent.
6. Configure a trusted server list. When identity assertion is selected for inbound requests, insert a pipe-separated (**|**) list of server administrator IDs to which this server can support identity token submission. For backwards compatibility, you can still use a comma-delimited list. However, if the server ID is a distinguished name (DN), then you must use a pipe-delimited (|) list because a comma delimiter does not work. If you choose to support any server sending an identity token, you can enter an asterisk (*) in this field. This action is called *presumed trust*. In this case, use SSL client certificate authentication between servers to establish the trust.
7. Configure session management. You can choose either *stateful* or *stateless* security. Performance is optimum when choosing stateful sessions. The first method request between a client and server is authenticated. All subsequent requests (or until the credential token expires) reuse the session information, including the credential. A client sends a context ID for subsequent requests. The context ID is scoped to the connection for uniqueness.

Results

When you finish configuring this panel, you have configured most of the information that a client gathers when determining what to send to this server. A client or server outbound configuration with this server inbound configuration, determines the security that is applied. When you know what clients send, the configuration is simple. However, if you have a diverse set of clients with differing security requirements, your server considers various layers of authentication.

For a J2EE application server, the authentication choice is usually either identity assertion or message layer because you want the identity of the originating client delegated downstream. You cannot easily delegate a client certificate using an SSL connection. It is acceptable to enable the transport layer

because additional server security, as the additional client certificate portion of the SSL handshake, adds some overhead to the overall SSL connection establishment.

What to do next

After you determine which type of authentication data this server might receive, you can determine what to select for outbound security. For more information, see Configuring Common Secure Interoperability Version 2 outbound authentication.

Common Secure Interoperability Version 2 inbound communications settings

Use this page to specify the features that a server supports for a client accessing its resources.

To view this administrative console page, complete the following steps:

1. Click **Security > Global security**.
2. From Authentication, click **RMI/IIOP security > CSIv2 inbound communications**.

Use common secure interoperability (CSI) inbound communications settings for configuring the type of authentication information that is contained in an incoming request or transport.

Authentication features include three layers of authentication that you can use simultaneously:

- **CSIv2 attribute layer**. The attribute layer might contain an identity token, which is an identity from an upstream server that already is authenticated. The identity layer has the highest priority, followed by the message layer, and then the transport layer. If a client sends all three, only the identity layer is used. The only way to use the SSL client certificate as the identity is if it is the only information that is presented during the request. The client picks up the interoperable object reference (IOR) from the namespace and reads the values from the tagged component to determine what the server needs for security.
- **CSIv2 transport layer**. The transport layer, which is the lowest layer, might contain a Secure Sockets Layer (SSL) client certificate as the identity.
- **CSIv2 message layer**. The message layer might contain a user ID and password or an authenticated token with an expiration.

Propagate security attributes:

Specifies support for security attribute propagation during login requests. When you select this option, the application server retains additional information about the login request, such as the authentication strength used, and retains the identity and location of the request originator.

If you do not select this option, the application server does not accept any additional login information to propagate to downstream servers.

Information	Value
Default:	Enabled

Important: When you use the replication services, ensure that the **Propagate security attributes** option is enabled.

Use identity assertion:

Specifies that identity assertion is a way to assert identities from one server to another during a downstream Enterprise JavaBeans (EJB) invocation.

This server does not authenticate the asserted identity again because it trusts the upstream server. Identity assertion takes precedence over all other types of authentication.

Identity assertion is performed in the attribute layer and is only applicable on servers. The principal determined at the server is based on precedence rules. If identity assertion is used, the identity is always derived from the attribute layer. If basic authentication is used without identity assertion, the identity is always derived from the message layer. Finally, if SSL client certificate authentication is performed without either basic authentication, or identity assertion, then the identity is derived from the transport layer.

The identity asserted is the invocation credential that is determined by the RunAs mode for the enterprise bean. If the RunAs mode is Client, the identity is the client identity. If the RunAs mode is System, the identity is the server identity. If the RunAs mode is Specified, the identity is the one specified. The receiving server receives the identity in an identity token and also receives the sending server identity in a client authentication token. The receiving server validates the sending server identity as a trusted identity through the Trusted Server IDs entry box. Enter a list of pipe-separated (|) principal names, for example, `serverid1|serverid2|serverid3`.

All identity token types map to the user ID field of the active user registry. For an ITTPrincipal identity token, this token maps one-to-one with the user ID fields. For an ITTDistinguishedName identity token, the value from the first equal sign is mapped to the user ID field. For an ITTCertChain identity token, the value from the first equal sign of the distinguished name is mapped to the user ID field.

When authenticating to an LDAP user registry, the LDAP filters determine how an identity of type ITTCertChain and ITTDistinguishedName get mapped to the registry. If the token type is ITTPrincipal, then the principal gets mapped to the UID field in the LDAP registry.

Information	Value
Default:	Disabled

Trusted identities:

Specifies the trusted identity that is sent from the sending server to the receiving server.

Specifies a pipe-separated (|) list of trusted server administrator user IDs, which are trusted to perform identity assertion to this server. For example, `serverid1|serverid2|serverid3`. The application server supports the comma (,) character as the list delimiter for backwards compatibility. The application server checks the comma character when the pipe character (|) fails to find a valid trusted server ID.

Use this list to decide whether a server is trusted. Even if the server is on the list, the sending server must still authenticate with the receiving server to accept the identity token of the sending server.

Information	Value
Data type:	String

Client certificate authentication:

Specifies that authentication occurs when the initial connection is made between the client and the server during a method request.

In the transport layer, Secure Sockets Layer (SSL) client certificate authentication occurs. In the message layer, basic authentication (user ID and password) is used. Client certificate authentication typically performs better than message layer authentication, but requires some additional setup. These additional steps involve verifying that the server trusts the signer certificate of each client to which it is connected. If the client uses a certificate authority (CA) to create its personal certificate, you only need the CA root certificate in the server signer section of the SSL trust file.

When the certificate is authenticated to a Lightweight Directory Access Protocol (LDAP) user registry, the distinguished name (DN) is mapped based on the filter that is specified when configuring LDAP. When the

certificate is authenticated to a local OS user registry, the first attribute of the distinguished name (DN) in the certificate, which is typically the common name, is mapped to the user ID in the registry.

The identity from client certificates is used only if no other layer of authentication is presented to the server.

Never Specifies that clients cannot attempt Secure Sockets Layer (SSL) client certificate authentication with this server.

Supported
Specifies that clients connecting to this server can authenticate using SSL client certificates. However, the server can invoke a method without this type of authentication. For example, anonymous or basic authentication can be used instead.

Required
Specifies that clients connecting to this server must authenticate using SSL client certificates before invoking the method.

Transport:

Specifies whether client processes connect to the server using one of its connected transports.

You can choose either Secure Sockets Layer (SSL), TCP/IP or both as the inbound transport that a server supports. If you specify TCP/IP, the server only supports TCP/IP and cannot accept SSL connections. If you specify SSL-supported, this server can support either TCP/IP or SSL connections. If you specify SSL-required, then any server communicating with this one must use SSL.

Note: This option is not available on the z/OS platform unless both Version 6.1 and earlier nodes exist in the cell.

TCP/IP
If you select **TCP/IP**, then the server opens a TCP/IP listener port only and all inbound requests do not have SSL protection.

SSL-required
If you select **SSL-required**, then the server opens an SSL listener port only and all inbound requests are received using SSL.

SSL-supported
If you select **SSL-supported**, then the server opens both a TCP/IP and an SSL listener port and most inbound requests are received using SSL.

Provide a fixed port number for the following ports. A zero port number indicates that a dynamic assignment is made at run time.

CSIV2_SSL_MUTUALAUTH_LISTENER_ADDRESS
CSIV2_SSL_SERVERAUTH_LISTENER_ADDRESS
SAS_SSL_SERVERAUTH_LISTENER_ADDRESS

Information	Value
Default:	SSL-required
Range:	TCP/IP, SSL Required, SSL-Supported

SSL settings:

Specifies a list of predefined SSL settings to choose from for inbound connection.

Information	Value
Data type:	String
Default: DefaultIIOPSSL	DefaultSSLSettings
Range:	Any SSL settings configured in the SSL Configuration Repertoire

Message layer authentication:

The following options are available for message layer authentication:

Never Specifies that this server cannot accept authentication using any of the following mechanisms selected.

Supported
Specifies that a client communicating with this server can authenticate using any of the following mechanisms selected. However, a method might be invoked without this type of authentication. For example, an anonymous or client certificate might be used instead.

Required
Specifies that clients communicating with this server must specify authentication information using of the following mechanisms selected for any method request.

Allow client to server authentication with::

Specifies client-to-server authentication using Kerberos, LTPA or Basic authentication.

The following options are available for client to server authentication:

Kerberos (KRB5)
Select to specify Kerberos as the authentication mechanism. You must first configure the Kerberos authentication mechanism. Read about Configuring Kerberos as the authentication mechanism using the administrative console for more information.

LTPA Select to specify the LTPA token authentication

Basic authentication
Basic authentication is Generic Security Services Username Password (GSSUP). This type of authentication typically involves sending a user ID and a password from the client to the server for authentication.

If you select **Basic Authentication** and **LTPA**, and the active authentication mechanism is LTPA, a user name, password, and LTPA tokens are accepted.

If you select **Basic Authentication** and **KRB5** and the active authentication mechanism is KRB5, a user name, password, Kerberos token and LTPA tokens are accepted.

If you do not select **Basic Authentication**, a user name and password are not accepted by the server.

Login configuration:

Specifies the type of system login configuration to use for inbound authentication.

You can add custom login modules by clicking **Security > Global security**. From Authentication, click **Java Authentication and Authorization Service > System logins**.

Stateful sessions:

Select this option to enable stateful sessions, which are used mostly for performance improvements.

The first contact between a client and server must fully authenticate. However, all subsequent contacts with valid sessions reuse the security information. The client passes a context ID to the server, and the ID is used to look up the session. The context ID is scoped to the connection, which guarantees uniqueness. Whenever the security session is not valid and the authentication retry is enabled, which is the default, the client-side security interceptor invalidates the client-side session and submits the request again without user awareness. This situation might occur if the session does not exist on the server; for example, the server failed and resumed operation. When this value is disabled, each method invocation must authenticate again.

Information	Value
Default:	Enabled

Trusted authentication realms - inbound:

Select this link to establish inbound trust for realms. Inbound authentication realm settings are not specific to CSIv2; you can also configure which realms to grant inbound trust to for multiple security domains.

Inbound authentication refers to the configuration that determines the type of accepted authentication for inbound requests. This authentication is advertised in the interoperable object reference (IOR) that the client retrieves from the name server.

Configuring Common Secure Interoperability Version 2 outbound communications

The following choices are available when configuring the Common Secure Interoperability Version 2 (CSIv2) outbound communications panel.

Before you begin

Outbound communications refers to the configuration that determines the type of authentication that is performed for outbound requests to downstream servers. Several *layers* or *methods* of authentication can occur. The downstream server inbound authentication configuration must support at least one choice made in this server outbound authentication configuration. If nothing is supported, the request might go outbound as unauthenticated. This situation does not create a security problem because the authorization runtime is responsible for preventing access to protected resources. However, if you choose to prevent an unauthenticated credential from going outbound, you might want to designate one of the authentication layers as required, rather than supported. If a downstream server does not support authentication, then when authentication is required, the method request fails to go outbound.

About this task

The following choices are available in the Common Secure Interoperability Version 2 (CSIv2) outbound communications panel. Remember that you are not required to complete these steps in the displayed order. Rather, these steps are provided to help you understand your choices for configuring outbound communications.

Procedure

- Select **Identity Assertion** (attribute layer). When selected, this server sends an identity token to a downstream server if the downstream server supports identity assertion. When an originating client authenticates to this server, the authentication information supplied is preserved in the outbound identity token. If the client authenticating to this server uses client certificate authentication, then the identity

token format is a certificate chain, containing the exact client certificate chain from the inbound socket. The same scenario is true for other mechanisms of authentication. Read theIdentity Assertion topic for more information.

- Select **User ID** and **Password** (message layer). This type of authentication is the most typical. The user ID and password (if BasicAuth credential) or authenticated token (if authenticated credential) are sent outbound to the downstream server if the downstream server supports message layer authentication in the inbound authentication panel. Refer to the Message Layer Authentication article for more information.
- Select **SSL Client certificate authentication** (transport layer). The main reason to enable outbound Secure Sockets Layer (SSL) client authentication from one server to a downstream server is to create a trusted environment between those servers. For delegating client credentials, use one of the two layers mentioned previously. However, you might want to create SSL personal certificates for all the servers in your domain, and only trust those servers in your SSL truststore file. No other servers or clients can connect to the servers in your domain, except at the tiers where you want them. This process can protect your enterprise bean servers from access by anything other than your servlet servers.

Example

Typically, the outbound authentication configuration is for an upstream server to communicate with a downstream server. Most likely, the upstream server is a servlet server and the downstream server is an Enterprise JavaBeans (EJB) server. On a servlet server, the client authentication that is performed to access the servlet can be one of many different types of authentication, including client certificate and basic authentication. When receiving basic authentication data, whether through a prompt login or a form-based login, the basic authentication information is typically authenticated to from a credential of the mechanism type that is supported by the server, such as the Lightweight Third Party Authentication (LTPA). When LTPA is the mechanism, a forwardable token exists in the credential. Choose the message layer (BasicAuth) authentication to propagate the client credentials. If the credential is created using a certificate login and you want to preserve sending the certificate downstream, you might decide to go outbound with identity assertion.

Common Secure Interoperability Version 2 outbound communications settings

Use this page to specify the features that a server supports when acting as a client to another downstream server.

To view this administrative console page, complete the following steps:

1. Click **Security > Global security**.
2. From Authentication, click **RMI/IIOP security > CSIv2 outbound communications**.

Authentication features include three layers of authentication that you can use simultaneously:

- **CSIv2 attribute layer**. The attribute layer might contain an identity token, which is an identity from an upstream server that already is authenticated. The identity layer has the highest priority, followed by the message layer, and then the transport layer. If a client sends all three, only the identity layer is used. The only way to use the SSL client certificate as the identity is if it is the only information that is presented during the request. The client picks up the interoperable object reference (IOR) from the namespace and reads the values from the tagged component to determine what the server needs for security.
- **CSIv2 transport layer**. The transport layer, which is the lowest layer, might contain a Secure Sockets Layer (SSL) client certificate as the identity.
- **CSIv2 message layer**. The message layer might contain a user ID and password or an authenticated token with an expiration.

Propagate security attributes:

Specifies to support security attribute propagation during login requests. When you select this option, the application server retains additional information about the login request, such as the authentication strength used, and retains the identity and location of the request originator.

If you do not select this option, the application server does not accept any additional login information to propagate to downstream servers.

Information	Value
Default:	Enabled

Important: When you use the replication services, ensure that the **Propagate security attributes** option is enabled.

Use identity assertion:

Specifies that identity assertion is a way to assert identities from one server to another during a downstream Enterprise JavaBeans (EJB) invocation.

This server does not authenticate the asserted identity again because it trusts the upstream server. Identity assertion takes precedence over all other types of authentication.

Identity assertion is performed in the attribute layer and is only applicable on servers. The principal determined at the server is based on precedence rules. If identity assertion is performed, the identity is always derived from the attribute layer. If basic authentication is used without identity assertion, the identity is always derived from the message layer. Finally, if SSL client certificate authentication is performed without either basic authentication, or identity assertion, then the identity is derived from the transport layer.

The identity asserted is the invocation credential that is determined by the RunAs mode for the enterprise bean. If the RunAs mode is Client, the identity is the client identity. If the RunAs mode is System, the identity is the server identity. If the RunAs mode is Specified, the identity is the one specified. The receiving server receives the identity in an identity token and also receives the sending server identity in a client authentication token. The receiving server validates the sending server identity as a trusted identity through the Trusted Server IDs entry box. Enter a list of pipe-separated (|) principal names, for example, `serverid1|serverid2|serverid3`.

All identity token types map to the user ID field of the active user registry. For an ITTPrincipal identity token, this token maps one-to-one with the user ID fields. For an ITTDistinguishedName identity token, the value from the first equal sign is mapped to the user ID field. For an ITTCertChain identity token, the value from the first equal sign of the distinguished name is mapped to the user ID field.

When authenticating to an LDAP user registry, the LDAP filters determine how an identity of type ITTCertChain and ITTDistinguishedName get mapped to the registry. If the token type is ITTPrincipal, then the principal gets mapped to the UID field in the LDAP registry.

Information	Value
Default:	Disabled

Use server-trusted identity:

Specifies the server identity that the application server uses to establish trust with the target server. The server identity can be sent using one of the following methods:

- A server ID and password when the server password is specified in the registry configuration.
- A server ID in a Lightweight Third Party Authentication (LTPA) token when the internal server ID is used.

For interoperability with application servers other than WebSphere Application Server, use one of the following methods:

- Configure the server ID and password in the registry.
- Select the **Server-trusted identity** option and specify the trusted identity and password so that an interoperable Generic Security Services Username Password (GSSUP) token is sent instead of an LTPA token.

Information	Value
Default:	Disabled

Specify an alternative trusted identity:

Specifies an alternative user as the trusted identity that is sent to the target servers instead of sending the server identity.

This option is recommended for identity assertion. The identity is automatically trusted when it is sent within the same cell and does not need to be in the trusted identities list within the same cell. However, this identity must be in the registry of the target servers in an external cell, and the user ID must be on the trusted identities list or the identity is rejected during trust evaluation.

Note: You must select Basic Authentication under the Message Layer authentication section to send an alternative trusted identity. If you do not select Basic Authentication, then choose the Server Identity instead.

Information	Value
Default:	Disabled

Trusted identity:

Specifies the trusted identity that is sent from the sending server to the receiving server.

If you specify an identity in this field, it can be selected on the panel for your configured user account repository. If you do not specify an identity, a Lightweight Third Party Authentication (LTPA) token is sent between the servers.

Specifies a pipe-separated (|) list of trusted server administrator user IDs, which are trusted to perform identity assertion to this server. For example, `serverid1|serverid2|serverid3`. The application server supports the comma (,) character as the list delimiter for backwards compatibility. The application server checks the comma character when the pipe character (|) fails to find a valid trusted server ID.

Use this list to decide whether a server is trusted. Even if the server is on the list, the sending server must still authenticate with the receiving server to accept the identity token of the sending server.

Password:

Specifies the password that is associated with the trusted identity.

Information	Value
Data type:	Text

Confirm password:

Confirms the password that is associated with the trusted identity.

Information	Value
Data type:	Text

Message layer authentication:

The following options are available for message layer authentication:

Never Specifies that this server cannot accept authentication using any of the mechanisms selected below.

Supported
Specifies that a client communicating with this server can authenticate using any of the mechanisms selected below. However, a method might be invoked without this type of authentication. For example, an anonymous or client certificate might be used instead.

Required
Specifies that clients communicating with this server must specify authentication information using of the mechanisms selected below for any method request.

Allow client to server authentication with::

Specifies client-to-server authentication using Kerberos, LTPA or Basic authentication.

The following options are available for client to server authentication:

Kerberos (KRB5)
Select to specify Kerberos as the authentication mechanism. You must first configure the Kerberos authentication mechanism. Read about Configuring Kerberos as the authentication mechanism using the administrative console for more information.

LTPA Select to configure and enable Lightweight Third-Party Authentication (LTPA) token authentication.

Basic authentication
Basic authentication is Generic Security Services Username Password (GSSUP). This type of authentication typically involves sending a user ID and a password from the client to the server for authentication.

If you select **Basic Authentication** and **LTPA**, and the active authentication mechanism is LTPA, the server goes with a downstream server with a user name, password or LTPA token.

If you select **Basic Authentication** and **KRB5**, and the active authentication mechanism is KRB5, the server goes with a downstream server with a user name, password, Kerberos token or LTPA token.

If you do not select **Basic Authentication**, the server does not go with a downstream server with a user name and password.

Transport:

Specifies whether client processes connect to the server using one of its connected transports.

You can choose to use either Secure Sockets Layer (SSL), TCP/IP or both as the inbound transport that a server supports. If you specify TCP/IP, the server only supports TCP/IP and cannot accept SSL connections. If you specify SSL-supported, this server can support either TCP/IP or SSL connections. If you specify SSL-required, then any server communicating with this one must use SSL.

For server-to-server communication, it is not enough to enable only the Transport layer. You must also enable either the Message layer or the Attribute layer.

TCP/IP
If you select **TCP/IP**, then the server opens a TCP/IP listener port only and all inbound requests do not have SSL protection.

SSL-required
If you select **SSL-required**, then the server opens an SSL listener port only and all inbound requests are received using SSL.

SSL-supported
If you select **SSL-supported**, then the server opens both a TCP/IP and an SSL listener port and most inbound requests are received using SSL.

Provide a fixed port number for the following ports. A zero port number indicates that a dynamic assignment is made at run time.

CSIV2_SSL_MUTUALAUTH_LISTENER_ADDRESS
CSIV2_SSL_SERVERAUTH_LISTENER_ADDRESS
SAS_SSL_SERVERAUTH_LISTENER_ADDRESS

Information	Value
Default:	SSL-Required
Range:	TCP/IP, SSL Required, SSL-Supported

SSL settings:

Specifies a list of predefined SSL settings to choose from for inbound connection.

Information	Value
Data type:	String
Default: DefaultIIOPSSL	DefaultSSLSettings
Range:	Any SSL settings configured in the SSL Configuration Repertoire

Client certificate authentication:

Specifies whether a client certificate from the configured keystore is used to authenticate to the server when the SSL connection is made between this server and a downstream server, provided that the downstream server supports client certificate authentication.

Typically, client certificate authentication has a higher performance than message layer authentication, but requires some additional setup. These additional steps include verifying that this server has a personal certificate and that the downstream server has the signer certificate of this server.

If you select client certificate authentication, the following options are available:

Never Specifies that this server does not attempt Secure Sockets Layer (SSL) client certificate authentication with downstream servers.

Supported
Specifies that this server can use SSL client certificates to authenticate to downstream servers. However, a method can be invoked without this type of authentication. For example, the server can use anonymous or basic authentication instead.

Required
Specifies that this server must use SSL client certificates to authenticate to downstream servers.

Information	Value
Default:	Enabled

Login configuration:

Specifies the type of system login configuration to use for inbound authentication.

You can add custom login modules by clicking **Security > Global security**. From Authentication, click **Java Authentication and Authorization Service > System logins**.

Stateful sessions:

Select this option to enable stateful sessions, which are used mostly for performance improvements.

The first contact between a client and server must fully authenticate. However, all subsequent contacts with valid sessions reuse the security information. The client passes a context ID to the server, and the ID is used to look up the session. The context ID is scoped to the connection, which guarantees uniqueness. Whenever the security session is not valid and the authentication retry is enabled, which is the default, the client-side security interceptor invalidates the client-side session and submits the request again without user awareness. This situation might occur if the session does not exist on the server, for example, the server failed and resumed operation. When this value is disabled, every method invocation must authenticate again.

Enable CSIv2 session cache limit:

Specifies whether to limit the size of the CSIv2 session cache.

When you enable this option, you must set values for the **Maximum cache size** and **Idle session timeout** options. When you do not enable this option, the CSIv2 session cache is not limited.

In previous versions of the application server, you might have set this value as the com.ibm.websphere.security.util.csiv2SessionCacheLimitEnabled custom property. In this product version, it is advisable to set this value using this administrative console panel and not as a custom property.

Information	Value
Default:	false

Maximum cache size:

Specify the maximum size of the session cache after which expired sessions are deleted from the cache.

Expired sessions are defined as sessions that are idle longer than the time that is specified in the **Idle session timeout** field. When you specify a value for the **Maximum cache size** field, consider setting its value between 100 and 1000 entries.

Consider specifying a value for this field if your environment uses Kerberos authentication and has a short clock skew for the configured key distribution center (KDC). In this scenario, a short clock skew is defined as less than 20 minutes. Consider increasing the value of this field if the small cache size causes the garbage collection to run so frequently that it impacts the performance of the application server.

In previous versions of the application server, you might have set this value as the com.ibm.websphere.security.util.csiv2SessionCacheMaxSize custom property. In this product version, it is advisable to set this value using this administrative console panel and not as a custom property.

This field only applies if you enable both the **Stateful sessions** and the **Enable CSIv2 session cache limit** options.

Information	Value
Default:	By default, a value is not set.
Range:	100 to 1000 entries

Idle session timeout:

This property specifies the time in milliseconds that a CSIv2 session can remain idle before being deleted. The session is deleted if you select the Enable CSIv2 session cache limit option and the value of the **Maximum cache size** field is exceeded.

This timeout value only applies if you enable both the **Stateful sessions** and the **Enable CSIv2 session cache limit** options. Consider decreasing the value for this field if your environment uses Kerberos authentication and has a short clock skew for the configured key distribution center (KDC). In this scenario, a short clock skew is defined as less than 20 minutes. A small clock skew can result in a larger number of rejected CSIv2 sessions. However, with a smaller value for the **Idle session timeout** field, the application server can clean out these rejected sessions more frequently and potentially reduce the resource shortages.

In previous versions of WebSphere Application Server, you might have set this value as the com.ibm.websphere.security.util.csiv2SessionCacheIdleTime custom property. In this product version, it is advisable to set this value using this administrative console panel and not as a custom property. If you previously set it as a custom property, the value was set in milliseconds and converted on this administrative console panel to seconds. On this administrative console panel, you must specify the value in seconds.

Information	Value
Default:	By default, a value is not set.
Range:	60 to 86,400 seconds

Custom outbound mapping:

Enables the use of custom Remote Method Invocation (RMI) outbound login modules.

The custom login module maps or completes other functions before the predefined RMI outbound call.

To declare a custom outbound mapping, complete the following steps:

1. Click **Security > Global security**.
2. From Authentication, click **Java Authentication and Authorization Service > System logins > New**.

Trusted authentication realms - outbound:

If the RMI/IIOP communication is across different realms, use this link to add outbound trusted realms.

The credential tokens are only sent to the realms that are trusted. In addition, the receiving server should trust this realm using the inbound trusted realms configuration to validate the LTPA token.

Configuring inbound transports

By using this configuration, you can configure a different transport for inbound security versus outbound security.

Before you begin

Inbound transports refer to the types of listener ports and their attributes that are opened to receive requests for this server. Both Common Secure Interoperability Specification, Version 2 (CSIv2) and Secure Authentication Service (SAS) have the ability to configure the transport.

Important: SAS is supported only between Version 6.0.x and previous version servers that have been federated in a Version 6.1 cell.

However, the following differences between the two protocols exist:
- CSIv2 is much more flexible than SAS, which requires Secure Sockets Layer (SSL); CSIv2 does not require SSL.
- SAS does not support SSL client certificate authentication, while CSIv2 does.
- CSIv2 can require SSL connections, while SAS only supports SSL connections.
- SAS always has two listener ports open: TCP/IP and SSL.
- CSIv2 can have as few as one listener port and as many as three listener ports. You can open one port for just TCP/IP or when SSL is required. You can open two ports when SSL is supported, and open three ports when SSL and SSL client certificate authentication is supported.

About this task

Complete the following steps to configure the Inbound transport panels in the administrative console:

Procedure

1. Click **Security > Global security**.
2. Under RMI/IIOP security, click **CSIv2 inbound communications**.
3. Under Transport, select **SSL-required**. You can choose to use either Secure Sockets Layer (SSL), TCP/IP or both as the inbound transport that a server supports. If you specify TCP/IP, the server only supports TCP/IP and cannot accept SSL connections. If you specify SSL-supported, this server can support either TCP/IP or SSL connections. If you specify SSL-required, then any server communicating with this one must use SSL.
4. Click **Apply**.
5. Consider fixing the listener ports that you configured.

   You complete this action in a different panel, but think about this action now. Most endpoints are managed at a single location, which is why they do not display in the Inbound transport panels. Managing end points at a single location helps you decrease the number of conflicts in your configuration when you assign the endpoints. The location for SSL end points is at each server. The following port names are defined in the End points panel and are used for Object Request Broker (ORB) security:
   - CSIV2_SSL_MUTUALAUTH_LISTENER_ADDRESS - CSIv2 Client Authentication SSL Port
   - CSIV2_SSL_SERVERAUTH_LISTENER_ADDRESS - CSIv2 SSL Port
   - SAS_SSL_SERVERAUTH_LISTENER_ADDRESS - SAS SSL Port
   - ORB_LISTENER_PORT - TCP/IP Port

   For an application server, click **Servers > Application servers >** *server_name*. Under Communications, click **Ports**. The Ports panel is displayed for the specified server.

   The Object Request Broker (ORB) on WebSphere Application Server uses a listener port for Remote Method Invocation over the Internet Inter-ORB Protocol (RMI/IIOP) communications, and is statically specified using configuration dialogs or during migration. If you are working with a firewall, you must specify a static port for the ORB listener and open that port on the firewall so that communication can pass through the specified port. The endPoint property for setting the ORB listener port is: ORB_LISTENER_ADDRESS.

   Complete the following steps using the administrative console to specify the ORB_LISTENER_ADDRESS port or ports.

a. Click **Servers > Application Servers >** *server_name*. Under Communications, click **Ports > New**.
b. Select **ORB_LISTENER_ADDRESS** from the **Port name** field in the Configuration panel.
c. Enter the IP address, the fully qualified Domain Name System (DNS) host name, or the DNS host name by itself in the **Host** field. For example, if the host name is `myhost`, the fully qualified DNS name can be `myhost.myco.com` and the IP address can be `155.123.88.201`.
d. Enter the port number in the **Port** field. The port number specifies the port for which the service is configured to accept client requests. The port value is used with the host name. Using the previous example, the port number might be 9000.

6. Click **Security > Global security**. Under RMI/IIOP security, click **CSIv2 inbound communications**. Select the SSL settings that are used for inbound requests from CSIv2 clients, and then click **Apply**. Remember that the CSIv2 protocol is used to inter-operate with previous releases. When configuring the keystore and truststore files in the SSL configuration, these files need the right information for inter-operating with previous releases of WebSphere Application Server.

Results

The inbound transport configuration is complete. With this configuration, you can configure a different transport for inbound security versus outbound security. For example, if the application server is the first server that is used by users, the security configuration might be more secure. When requests go to back-end enterprise bean servers, you might lessen the security for performance reasons when you go outbound. With this flexibility you can design the right transport infrastructure to meet your needs.

What to do next

When you finish configuring security, perform the following steps to save, synchronize, and restart the servers:
1. Click **Save** in the administrative console to save any modifications to the configuration.
2. Stop and restart all servers, when synchronized.

Common Secure Interoperability Version 2 transport inbound settings

Use this page to specify which listener ports to open and which Secure Sockets Layer (SSL) settings to use. These specifications determine which transport a client or upstream server uses to communicate with this server for incoming requests.

To view this administrative console page, complete the following steps:
1. Click **Security > Global security**.
2. Under Authentication, click **RMI/IIOP security > CSIv2 inbound transport**.

Transport:

Specifies whether client processes connect to the server using one of its connected transports.

You can choose to use either Secure Sockets Layer (SSL), TCP/IP or both as the inbound transport that a server supports. If you specify TCP/IP, the server only supports TCP/IP and cannot accept SSL connections. If you specify SSL-supported, this server can support either TCP/IP or SSL connections. If you specify SSL-required, then any server communicating with this one must use SSL.

If you specify SSL-supported or SSL-required, decide which set of SSL configuration settings you want to use for the inbound configuration. This decision determines which key file and trust file are used for inbound connections to this server.

TCP/IP

If you select **TCP/IP**, then the server opens a TCP/IP listener port only and all inbound requests do not have SSL protection.

SSL-required
If you select **SSL-required**, then the server opens an SSL listener port only and all inbound requests are received using SSL.

Important: SAS is supported only between Version 6.0.x and previous version servers that have been federated in a Version 6.1 cell.

SSL-supported
If you select **SSL-supported**, then the server opens both a TCP/IP and an SSL listener port and most inbound requests are received using SSL.

Provide a fixed port number for the following ports. A zero port number indicates that a dynamic assignment is made at runtime.

CSIV2_SSL_MUTUALAUTH_LISTENER_ADDRESS
CSIV2_SSL_SERVERAUTH_LISTENER_ADDRESS
SAS_SSL_SERVERAUTH_LISTENER_ADDRESS

Information	Value
Default:	SSL Required
Range:	TCP/IP, SSL Required, SSL-Supported

SSL settings:

Specifies a list of predefined SSL settings to choose from for inbound connections.

These settings are configured at the SSL Repertoire panel. To access the SSL Repertoire panel, complete the following steps:

1. Clicking **Security > SSL certificate and key management**.
2. Under configuration settings, click **Manage endpoint security configurations and trust zones**.
3. Expand Inbound and click *inbound_configuration*.
4. Under Related items, click **SSL configurations**.

Information	Value
Data type:	String
Default: DefaultIIOPSSL	DefaultSSLSettings
Range:	Any SSL settings configured in the SSL Configuration Repertoire

Centrally managed:

Specifies that the selection of an SSL configuration is based upon the outbound topology view for the Java Naming and Directory Interface (JNDI) platform.

Centrally managed configurations support one location to maintain SSL configurations rather than spreading them across the configuration documents.

Information	Value
Default:	Enabled

Use specific SSL alias:

Specifies the SSL configuration alias to use for LDAP outbound SSL communications.

This option overrides the centrally managed configuration for the JNDI platform.

z/OS SSL settings:

Specifies a list of predefined Secure Sockets Layer (SSL) settings for inbound connections. Configure these settings on the SSL panel by clicking Secure communications on the administrative console.

Secure Authentication Service inbound transport settings

Use this page to specify transport settings for connections that are accepted by this server using the Secure Authentication Service (SAS) authentication protocol. The SAS protocol is used to communicate securely to enterprise beans with previous releases of the application server.

To view this administrative console page, complete the following steps:

1. Click **Security > Global security**.
2. Under Authentication, expand RMI/IIOP security and click **SAS inbound transport**.

Attention: The panel associated with this article displays only when you have a Version 6.1 server in your environment. SAS is supported only between Version 6.0.x and previous version servers that have been federated in a Version 6.1 cell.

SSL Settings:

Specifies a list of predefined SSL settings to choose from for inbound connections.

These settings are configured on the Secure Sockets Layer (SSL) configuration panel. To access the SSL configuration panel, complete the following steps:

1. Click **Security > SSL certificate and key management > Manage endpoint security configurations and trust zones**.
2. Expand Inbound > *configuration_name*.
3. Under Related Items, click **SSL configurations**.

Information	Value
Data type:	String
Default:	DefaultSSLSettings

Configuring outbound transports

By using this configuration, you can configure a different transport for inbound security versus outbound security.

Before you begin

Outbound transports refers to the transport that is used to connect to a downstream server. When you configure the outbound transport, consider the transports that the downstream servers support. If you are considering Secure Sockets Layer (SSL), also consider including the signers of the downstream servers in this server truststore file for the handshake to succeed.

When you select an SSL configuration, that configuration points to keystore and truststore files that contain the necessary signers.

If you configured client certificate authentication for this server by completing the following steps, then the downstream servers contain the signer certificate belonging to the server personal certificate:

1. Click **Security > Global security**.
2. Under RMI/IIOP security, click **CSIv2 outbound communications**.

About this task

Complete the following steps to configure the outbound transport panels.

Procedure

1. Select the type of transport and the SSL settings by clicking **Security > Global security**. Under RMI/IIOP security, click **CSIv2 outbound communications**. By selecting the type of transport, you choose the transport to use when connecting to downstream servers. The downstream servers support the transport that you choose. If you choose **SSL-Supported**, the transport that is used is negotiated during the connection. If both the client and server support SSL, always select the **SSL-Supported** option unless the request is considered a special request that does not require SSL, such as if an object request broker (ORB) is a request.
2. Select the **SSL required** option if you want to use Secure Sockets Layer communications with the outbound transport.

   If you select the **SSL required** option or the **SSL supported** option, you can select either the **Centrally managed** or **Use specific SSL alias** option.

   Centrally managed
   Enables you to specify an SSL configuration for particular scope such as the cell, node, server, or cluster in one location. To use the **Centrally managed** option, you must specify the SSL configuration for the particular set of endpoints. The Manage endpoint security configurations and trust zones panel displays all of the inbound and outbound endpoints that use the SSL protocol. If you expand the Inbound or Outbound section of the panel and click the name of a node, you can specify an SSL configuration that is used for every endpoint on that node. For an outbound transport, you can override the inherited SSL configuration by specifying an SSL configuration for a particular endpoint. To specify an SSL configuration for an outbound transport, click **Security > SSL certificate and key management > Manage endpoint security configurations and trust zones** and expand **Outbound**.

   Use specific SSL alias
   Select the **Use specific SSL alias** option if you intend to select one of the SSL configurations in the menu below the option.

   The default is *DefaultSSLSettings*. To modify or create a new SSL configuration, complete the steps described in “Creating a Secure Sockets Layer configuration” on page 713.
3. Click **Apply**.

Results

The outbound transport configuration is complete. With this configuration, you can configure a different transport for inbound security versus outbound security. For example, if the application server is the first server used by end users, the security configuration might be more secure. When requests go to back-end enterprise beans servers, you might consider less security for performance reasons when you go outbound. With this flexibility you can design a transport infrastructure that meets your needs.

What to do next

When you finish configuring security, perform the following steps to save, synchronize, and restart the servers.

- Click **Save** in the administrative console to save any modifications to the configuration.
- Stop and restart all servers, after synchronization.

Common Secure Interoperability Version 2 outbound transport settings

Use this page to specify which transports and Secure Sockets Layer (SSL) settings this server uses when communicating with downstream servers for outbound requests.

To view this administrative console page, complete the following steps:

1. Click **Security > Global security**.
2. Under Authentication, click **RMI/IIOP security > CSIv2 outbound transport**.

You also can view this administrative console by completing the following steps:
1. Click **Servers** > **Server Types** > **WebSphere application servers** > ***server_name***.
2. Under Security, click **Security Domain**, and then expand the **RMI/IIOP security** section under Authentication.
3. Click **CSIv2 outbound communications**.

Transport:

Specifies whether the client processes connect to the server using one of the server-connected transports.

You can choose to use either SSL, TCP/IP, or Both as the outbound transport that a server supports. If you specify `TCP/IP`, the server supports only TCP/IP and cannot initiate SSL connections with downstream servers. If you specify `SSL-supported`, this server can initiate either TCP/IP or SSL connections. If you specify `SSL-required`, this server must use SSL to initiate connections to downstream servers. When you do specify SSL, decide which set of SSL configuration settings you want to use for the outbound configuration.

This decision determines which keyfile and trustfile to use for outbound connections to downstream servers.

Consider the following options:

TCP/IP
: If you select this option, the server opens TCP/IP connections with downstream servers only.

SSL-required
: If you select this option, the server opens SSL connections with downstream servers.

SSL-supported
: If you select this option, the server opens SSL connections with any downstream server that supports them and opens TCP/IP connections with any downstream servers that do not support SSL.

Information	Value
Default:	SSL-supported
Range:	TCP/IP, SSL-required, SSL-supported

SSL settings:

Specifies a list of predefined SSL settings for outbound connections. These settings are configured at the SSL Configuration Repertoires panel.

To access the panel, complete the following steps:
1. Click **Security > SSL certificate and key management**.
2. Under Configuration settings, click **Manage endpoint security configurations and trust zones**.
3. Expand Outbound > *outbound_configuration_name*.
4. Under Related items, click **SSL configurations**.

Information	Value
Data type:	String
Range:	Any SSL settings that are configured in the SSL Configuration Repertoires panel

Note: This field is available only if a Version 6.1 server exists in your environment.

SSL enabled:

Specifies whether secure socket communication is enabled to the server.

Centrally managed:

Specifies that the selection of an SSL configuration is based upon the outbound topology view for the Java Naming and Directory Interface (JNDI) platform.

Centrally managed configurations support one location to maintain SSL configurations rather than spreading them across the configuration documents.

Information	Value
Default:	Enabled

Use specific SSL alias:

Specifies the SSL configuration alias that you want to use for outbound SSL communications.

This option overrides the centrally managed configuration for the JNDI (LDAP) protocol.

Secure Authentication Service outbound transport settings

Use this page to specify transport settings for connections that are accepted by this server using the Secure Authentication Service (SAS) authentication protocol.

To view this administrative console page, complete the following steps:

1. Click **Security > Global security**.
2. Under Authentication, expand RMI/IIOP security and click **SAS outbound transport**.

Attention: The panel associated with this article displays only when you have a Version 6.1 server in your environment.

SSL settings:

Specifies a list of predefined Secure Sockets Layer (SSL) settings to choose from for outbound connections.

These settings are configured on the SSL configuration panel. To access the SSL configuration panel, complete the following steps:

1. Click **Security > SSL certificate and key management > Manage endpoint security configurations and trust zones**.
2. Expand Outbound > *configuration_name*.
3. Under Related Items, click **SSL configurations**.

Information	Value
Data type:	String
Default:	DefaultSSLSettings

Configuring inbound messages

You can use the administrative console to configure inbound messages for CSIv2.

Procedure

1. In the administrative console, click **Security > Global security**.
2. Under Authentication, expand **RMI/HOP security**.
3. Click **CSIv2 inbound communication**.
4. Optional: Click **Propagate security attributes** or **Use identity assertion**. The **Propagate security attributes** option enables support for security attribute propagation during login requests. When you select this option, the application server retains additional information about the login request, such as the authentication strength used, and retains the identity and location of the request originator.

   The **Use identity assertion** option specifies that identity assertion is a way to assert identities from one server to another during a downstream Enterprise JavaBeans (EJB) invocation.
5. Under CSIv2 Message layer authentication, select **Supported**, **Never** or **Required**.

   Never Specifies that this server cannot accept an authentication mechanism that you select under **Allow client to server authentication with:**.

   Supported
   Specifies that clients communicating with this server can specify an authentication mechanism that you select under **Allow client to server authentication with:**. However, a method might be invoked without this type of authentication. For example, an anonymous or client certificate might be used instead.

   Required
   Specifies that clients communicating with this server must specify an authentication mechanism that you select under **Allow client to server authentication with:**.
6. Under **Allow client to server authentication with:**, select **Kerberos**, **LTPA** and or **Basic authentication**. You can optionally select:

   Kerberos
   Select to enable authentication using the Kerberos token.

   LTPA Select to enable authentication using the Lightweight Third-Party Authentication (LTPA) token.

   Basic authentication
   This type of authentication typically involves sending a user ID and a password from the client to the server for authentication. This is also know as Generic Security Services Username Password (GSSUP).

   This authentication also involves delegating a credential token from an already authenticated credential, provided the credential type is forwardable; for example, LTPA.

   If you select **supported** under **CSIv2 Message layer authentication**, and check **KRB5** and **LTPA** under **Allow client to server authentication with:**, then the server does not accept the user name and password.
7. Click **OK.**

Results

You have now configured messages for CSIv2 inbound.

Configuring outbound messages

You can use the administrative console to configure outbound messages for CSIv2.

Procedure

1. In the administrative console, click **Security > Global security**.
2. Under Authentication, expand **RMI/HOP security**.
3. Click **CSIv2 outbound communication**.

4. Optional: Click **Propagate security attributes** or **Use identity assertion**. The **Propagate security attributes** option enables support for security attribute propagation during login requests. When you select this option, the application server retains additional information about the login request, such as the authentication strength used, and retains the identity and location of the request originator.

   The **Use identity assertion** option specifies that identity assertion is a way to assert identities from one server to another during a downstream Enterprise JavaBeans (EJB) invocation.

   The **Use server trusted identity** option specifies the server identity that the application server uses to establish trust with the target server.

   The **Specify an alternative trusted identity** option enables you to specify an alternative user as the trusted identity that is sent to the target servers instead of sending the server identity. If you select this option you must provide the name of the trusted identity and the password that is associated with the trusted identity.

   Note: You must select Basic Authentication under the Message Layer authentication section to send an alternative trusted identity. If you do not select Basic Authentication, then choose the Server Identity instead.

5. Under CSIv2 Message layer authentication, select **Supported**, **Never** or **Required**.

   Never Specifies that this server cannot accept an authentication mechanism that you select under **Allow client to server authentication with:**.

   Supported
   Specifies that clients communicating with this server can specify an authentication mechanism that you select under **Allow client to server authentication with:**. However, a method might be invoked without this type of authentication. For example, an anonymous or client certificate might be used instead.

   Required
   Specifies that clients communicating with this server must specify an authentication mechanism that you select under **Allow client to server authentication with:**.

6. Under **Allow client to server authentication with:**, select **Kerberos**, **LTPA** and or **Basic authentication**. You can optionally select:.

   Kerberos
   Select to enable authentication using the Kerberos token.

   LTPA Select to enable authentication using the Lightweight Third-Party Authentication (LTPA) token.

   Basic authentication
   This type of authentication typically involves sending a user ID and a password from the client to the server for authentication. This is also know as Generic Security Services Username Password (GSSUP).

   This authentication also involves delegating a credential token from an already authenticated credential, provided the credential type is forwardable; for example, LTPA.

   If you select **supported** under **CSIv2 Message layer authentication**, and check **KRB5** and **LTPA** under **Allow client to server authentication with:**, then the server does not accept the user name and password.

7. Optional: Select **Custom outbound mapping**. This option enables the use of custom Remote Method Invocation (RMI) outbound login modules.

Results

You have now configured messages for CSIv2 outbound.

Common Secure Interoperability Version 2 and Security Authentication Service (SAS) client configuration

A secure Java client requires configuration properties to determine how to perform security with a server.

These configuration properties are typically put into a properties file somewhere on the client system and referenced by specifying the following system property on the command line of the Java client. For example, this property accepts any valid web address.

-Dcom.ibm.CORBA.ConfigURL=file:*profile_root*/properties/sas.client.props

When this file is processed by the Object Request Broker (ORB), security can be enabled between the Java client and the target server.

If any syntax problems exist with the ConfigURL property and the sas.client.props file is not found, the Java client proceeds to connect insecurely. Errors display indicating the failure to read the ConfigURL property. Typically the problem is related to having two slashes after file, which is not valid.

Use the following properties to configure the Secure Authentication Service (SAS) and CSIv2 authentication protocols:

- "Security Authentication Service authentication protocol client settings" on page 514

Important: SAS is supported only between Version 6.0.x and previous version servers that have been federated in a Version 6.1 cell.

Authentication protocol settings for a client configuration

You can use settings in the sas.client.props file to configure Security Authentication Service (SAS) and Common Secure Interoperability Version 2 (CSIv2) clients.

Use the following settings in the *app_server_root*/properties/sas.client.props file to configure SAS and CSIv2 clients.

Important: SAS is supported only between Version 6.0.x and previous version servers that have been federated in a Version 6.1 cell.

Note: The sas.client.props file for WebSphere Application Server Version 8.5 contains some new properties that support BasicAuth and Kerberos, such as:

```
com.ibm.IPC.authenticationTarget=BasicAuthcom.ibm.IPC.loginUserid=
com.ibm.IPC.loginPassword=com.ibm.IPC.loginSource=promptcom.ibm.IPC.krb5Service=WAScom.ibm.IPC.krb5Ccache
```

com.ibm.CORBA.securityEnabled:

Use to determine if security is enabled for the client process.

Table 45. com.ibm.CORBA.securityEnabled. This table describes the com.ibm.CORBA.securityEnabled setting.

Setting	Value
Data Type	Boolean
Default	True
Valid values	True or false

com.ibm.CSI.protocol:

Use to determine which authentication protocols are active.

The client can configure protocols of `ibm`, `csiv2` or `both` as active. The only possible values for an authentication protocol are `ibm`, `csiv2` and `both`. Do not use `sas` for the value of an authentication protocol. This restriction applies to both client and server configurations. The following list provides information about using each of these protocol options:

ibm Use this authentication protocol option when you are communicating with WebSphere Application Server Version 4.x or earlier servers.

csiv2 Use this authentication protocol option when you are communicating with WebSphere Application Server Version 5 or later servers because the SAS interceptors are not loaded and running for each method request.

both Use this authentication protocol option for interoperability between WebSphere Application Server Version 4.x or earlier servers and WebSphere Application Server Version 5 or later servers. Typically, specifying `both` provides greater interoperability with other servers.

Table 46. com.ibm.CSI.protocol. This table describes the com.ibm.CSI.protocol setting.

Setting	Value
Data type	String
Default	Both
Valid values	ibm, csiv2, both

com.ibm.CORBA.authenticationTarget:

Use to determine the type of authentication mechanism for sending security information from the client to the server.

If basic authentication is specified, the user ID and password are sent to the server. Using the Secure Sockets Layer (SSL) transport with this type of authentication is recommended; otherwise, the password is not encrypted. The target server must support the specified authentication target.

Table 47. com.ibm.CORBA.authenticationTarget. This table describes the com.ibm.CORBA.authenticationTarget setting.

Setting	Value
Data type	String
Default	BasicAuth
Valid values	BasicAuth, KRB5

com.ibm.CORBA.validateBasicAuth:

Use to determine if the user ID and password get validated immediately after the login data is entered when the authenticationTarget property is set to `BasicAuth`.

In previous releases, BasicAuth logins validated only with the initial method request. During the first request, the user ID and password are sent to the server. This request is the first time that the client can notice an error, if the user ID or password is incorrect. The validateBasicAuth method is specified and the validation of the user ID and password occurs immediately to the security server.

For performance reasons, you might want to disable this property if you do not want to verify the user ID and password immediately. If the client program can wait, it is better to have the initial method request flow to the user ID and password. However, program logic might not be this simple because of error handling considerations.

Table 48. com.ibm.CORBA.validateBasicAuth. This table describes the com.ibm.CORBA.validateBasicAuth setting.

Setting	Value
Data type	Boolean

Table 48. com.ibm.CORBA.validateBasicAuth (continued). This table describes the com.ibm.CORBA.validateBasicAuth setting.

Setting	Value
Default	True
Valid values	True, False

com.ibm.CORBA.authenticationRetryEnabled:

Use to specify that a failed login attempt is retried. This property determines if a retry occurs for other errors, such as stateful sessions that are not found on a server or validation failures at the server because of an expiring credential.

The minor code in the exception that is returned to a client determines which errors are retried. The number of retry attempts is dependent upon the com.ibm.CORBA.authenticationRetryCount property.

Table 49. com.ibm.CORBA.authenticationRetryEnabled. This table describes the com.ibm.CORBA.authenticationRetryEnabled setting.

Setting	Value
Data type	Boolean
Default	True
Valid values	True, False

com.ibm.CORBA.authenticationRetryCount:

Use to specify the number of retries that occur until either a successful authentication occurs or the maximum retry value is reached.

When the maximum retry value is reached, the authentication exception is returned to the client.

Table 50. com.ibm.CORBA.authenticationRetryCount. This table describes the com.ibm.CORBA.authenticationRetryCount setting.

Setting	Value
Data type	Integer
Default	3
Range	1-10

com.ibm.CORBA.loginSource:

Use to specify how the request interceptor attempts to log in if it does not find an invocation credential already set.

This property is valid only if message layer authentication occurs. If only transport layer authentication occurs, this property is ignored. When specifying properties, the following two additional properties must be defined:

- com.ibm.CORBA.loginUserid
- com.ibm.CORBA.loginPassword

When performing a programmatic login, it is not necessary to specify `none` as the login source. The request fails if a credential is set as the invocation credential during a method request.

Important: For the distributed platform, you can choose to NOT edit the properties file, sas.client.props, but set the loginSource property as follows: com.ibm.CORBA.loginSource=none

When you set com.ibm.CORBA.loginSource=none for a remote method invocation (RMI) connection, whether using scripting with wsadmin or from other clients, you have to perform a programmatic logon because the logged-in user's credentials are not inherited. You must specify user and/or password at the command line.

Table 51. com.ibm.CORBA.loginSource. This table describes the com.ibm.CORBA.loginSource setting.

Setting	Value
Data type	String
Default	Prompt
Valid values	Prompt, key file, stdin, none, properties

com.ibm.CORBA.loginUserid:

Use to specify the user ID when a properties login is configured and message layer authentication occurs.

This property is valid only when `com.ibm.CORBA.loginSource=properties`. Also set the com.ibm.CORBA.loginPassword property.

Table 52. com.ibm.CORBA.loginUserid. This table describes the com.ibm.CORBA.loginUserid setting.

Setting	Value
Data type	String
Range	Any string that is appropriate for a user ID in the configured user registry of the server.

com.ibm.CORBA.loginPassword:

Use to specify the password when a properties login is configured and message layer authentication occurs.

This property is valid only when `com.ibm.CORBA.loginSource=properties`. Also set the com.ibm.CORBA.loginUserid property.

Table 53. com.ibm.CORBA.loginPassword. This table describes the com.ibm.CORBA.loginPassword setting.

Setting	Value
Data type	String
Range	Any string that is appropriate for a password in the configured user registry of the server.

com.ibm.CORBA.keyFileName:

Use to specify the key file that is used to log in.

A key file is a file that contains a list of realm, user ID, and password combinations that a client uses to log into multiple realms. The realm that is used is the one found in the interoperable object reference (IOR) for the current method request. The value of this property is used when the `com.ibm.CORBA.loginSource=key file` is used.

Table 54. com.ibm.CORBA.keyFileName. This table describes the com.ibm.CORBA.keyFileName setting.

Setting	Value
Data type	String
Default	C;/WebSphere/AppServer/properties/wsserver.key
Range	Any fully qualified path and file name of a WebSphere Application Server key file.

com.ibm.CORBA.loginTimeout:

Use to specify the length of time that the login prompt stays available before it is considered a failed login.

Table 55. com.ibm.CORBA.loginTimeout. This table describes the com.ibm.CORBA.loginTimeout setting.

Setting	Value
Data type	Integer
Units	Seconds
Default	300 (5 minute intervals)
Range	0 - 600 (10 minute intervals)

com.ibm.CORBA.securityEnabled:

Use to determine if security is enabled for the client process.

Table 56. com.ibm.CORBA.securityEnabled. This table describes the com.ibm.CORBA.securityEnabled setting.

Setting	Value
Data type	Boolean
Default	True
Range	True, False

Security Authentication Service authentication protocol client settings

In addition to those properties which are valid for both Security Authentication Service (SAS) and Common Secure Interoperability Version 2 (CSIv2), this article documents properties which are valid only for the SAS authentication protocol.

Important: SAS is supported only between Version 6.0.x and previous version servers that have been federated in a Version 6.1 cell.

com.ibm.CORBA.standardPerformQOPModels:

Specifies the strength of the ciphers when making a Secure Sockets Layer (SSL) connection.

Information	Value
Data type:	String
Default:	High
Range	Low, Medium, High

Example 1: Configuring basic authentication and identity assertion

This example presents a pure Java client, C, that accesses a secure enterprise bean on server, S1, through user bob. The following steps take you through the configuration of C, S1, and S2.

About this task

The enterprise bean code on S1 accesses another enterprise bean on server, S2. This configuration uses identity assertion to propagate the identity of bob to the downstream server, S2. S2 trusts that bob already is authenticated by S1 because it trusts S1. To gain this trust, the identity of S1 also flows to S2 simultaneously and S2 validates the identity by checking the trustedPrincipalList list to verify that it is a valid server principal. S2 also authenticates S1.

Procedure

1. Configure the client C for message layer authentication with a Secure Sockets Layer (SSL) transport.
   a. Point the client to the `sas.client.props` file.

Use the `com.ibm.CORBA.ConfigURL=file:/C:/was/properties/sas.client.props` property. All further configuration involves setting properties within this file.

b. Enable SSL.

In this case, SSL is supported but not required:
`com.ibm.CSI.performTransportAssocSSLTLSSupported=true,`
`com.ibm.CSI.performTransportAssocSSLTLSRequired=false`

c. Enable client authentication at the message layer.

In this case, client authentication is supported but not required:
`com.ibm.CSI.performClientAuthenticationRequired=false,`
`com.ibm.CSI.performClientAuthenticationSupported=true`

d. Use all of the remaining defaults in the `sas.client.props` file.

2. Configure the server, S1.

In the administrative console, server S1 is configured for incoming requests to support message-layer client authentication and incoming connections to support SSL without client certificate authentication. Server S1 is configured for outgoing requests to support identity assertion.

a. Configure S1 for incoming connections.
   1) Disable identity assertion.
   2) Enable user ID and password authentication.
   3) Enable SSL.
   4) Disable SSL client certificate authentication.

b. Configure S1 for outgoing connections.
   1) Enable identity assertion.
   2) Disable user ID and password authentication.
   3) Enable SSL.
   4) Disable SSL client certificate authentication.

3. Configure the server, S2.

In the administrative console, server S2 is configured for incoming requests to support identity assertion and to accept SSL connections. Complete the following steps to configure incoming connections. Configuration for outgoing requests and connections are not relevant for this example.

a. Enable identity assertion.
b. Disable user ID and password authentication.
c. Enable SSL.
d. Disable SSL client authentication.

Example 2: Configuring basic authentication, identity assertion, and client certificates

This example is the same as example 1, except for the interaction from client C2 to server S2. Therefore, the configuration of example 1 still is valid, but you have to modify server S2 slightly and add a configuration for client C2. The configuration is not modified for C1 or S1.

About this task

Procedure

1. Configure client C2 for transport layer authentication (Secure Sockets Layer (SSL) client certificates).

a. Point the client to the `sas.client.props` file.

Use the `com.ibm.CORBA.ConfigURL=file:/C:/was/properties/sas.client.props` property. All further configuration involves setting properties within this file.

b. Enable SSL.

In this case, SSL is supported but not required:

```
com.ibm.CSI.performTransportAssocSSLTLSSupported=true,
com.ibm.CSI.performTransportAssocSSLTLSRequired=false
```

c. Disable client authentication at the message layer.

```
com.ibm.CSI.performClientAuthenticationRequired=false,
com.ibm.CSI.performClientAuthenticationSupported=false
```

d. Enable client authentication at the transport layer where it is supported, but not required.

```
com.ibm.CSI.performTLClientAuthenticationRequired=false,
com.ibm.CSI.performTLClientAuthenticationSupported=true
```

2. Configure the server, S2.

In the administrative console, server S2 is configured for incoming requests to SSL client authentication and identity assertion. Configuration for outgoing requests is not relevant for this example.

You can mix and match these configuration options. However, a precedence exists as to which authentication features become the identity in the received credential:

a. Identity assertion
b. Message-layer client authentication (basic authentication or token)
c. Transport-layer client authentication (SSL certificates)

a. Enable identity assertion.
b. Disable user ID and password authentication.
c. Enable SSL.
d. Enable SSL client authentication.

Example 3: Configuring client certificate authentication and RunAs system

This example presents a pure Java client, C, accessing a secure enterprise bean on S1.

About this task

C authenticates to S1 using Secure Sockets Layer (SSL) client certificates. S1 maps the common name of the distinguished name (DN) in the certificate to a user in the local registry. The user in this case is bob. The enterprise bean code on S1 accesses another enterprise bean on S2. Because the RunAs mode is `system`, the invocation credential is set as server1 for any outbound requests.

Procedure

1. Configure client C for transport layer authentication (SSL client certificates).

a. Point the client to the `sas.client.props` file.

Use the `com.ibm.CORBA.ConfigURL=file:/C:/was/properties/sas.client.props` property. All further configuration involves setting properties within this file.

b. Enable SSL.

In this case, SSL is supported but not required:

```
com.ibm.CSI.performTransportAssocSSLTLSSupported=true,
com.ibm.CSI.performTransportAssocSSLTLSRequired=false
```

c. Disable client authentication at the message layer.
`com.ibm.CSI.performClientAuthenticationRequired=false,`
`com.ibm.CSI.performClientAuthenticationSupported=false`

d. Enable client authentication at the transport layer. It is supported, but not required.
`com.ibm.CSI.performTLClientAuthenticationRequired=false,`
`com.ibm.CSI.performTLClientAuthenticationSupported=true`

2. Configure the S1 server. In the administrative console, S1 is configured for incoming connections to support SSL with client certificate authentication. The S1 server is configured for outgoing requests to support message layer client authentication.
   a. Configure S1 for incoming connections.
      1) Disable identity assertion.
      2) Disable user ID and password authentication.
      3) Enable SSL.
      4) Enable SSL client certificate authentication.
   b. Configure S1 for outgoing connections.
      1) Disable identity assertion.
      2) Disable user ID and password authentication.
      3) Enable SSL.
      4) Enable SSL client certificate authentication.
3. Configure the S2 server.

   In the administrative console, the S2 server is configured for incoming requests to support message layer authentication over SSL. Configuration for outgoing requests is not relevant for this scenario.
   a. Disable identity assertion.
   b. Enable user ID and password authentication.
   c. Enable SSL.
   d. Disable SSL client authentication.

Example 4: Configuring TCP/IP transport using a virtual private network

This scenario illustrates the ability to choose TCP/IP as the transport when it is appropriate. In some cases, when two servers are on the same virtual private network (VPN), it can be appropriate to select TCP/IP as the transport for performance reasons because the VPN already encrypts the message.

About this task

Procedure

1. Configure client C for message layer authentication with an Secure Sockets Layer (SSL) transport.
   a. Point the client to the `sas.client.props` file.

      Use the `com.ibm.CORBA.ConfigURL=file:/C:/was/properties/sas.client.props` property. All further configuration involves setting properties within this file.
   b. Enable SSL.

      In this case, SSL is supported but not required.`com.ibm.CSI.performTransportAssocSSLTLSSupported=true,`
      `com.ibm.CSI.performTransportAssocSSLTLSRequired=false`
   c. Enable client authentication at the message layer. In this case, client authentication is supported but not required. `com.ibm.CSI.performClientAuthenticationRequired=false,`
      `com.ibm.CSI.performClientAuthenticationSupported=true`

d. Use the remaining defaults in the `sas.client.props` file.

2. Configure the S1 server. In the administrative console, the S1 server is configured for incoming requests to support message-layer client authentication and incoming connections to support SSL without client certificate authentication. The S1 server is configured for outgoing requests to support identity assertion.

   It is possible to enable SSL for inbound connections and disable SSL for outbound connections. The same is true in reverse.

   a. Configure S1 for incoming connections.
      1) Disable identity assertion.
      2) Enable user ID and password authentication.
      3) Enable SSL.
      4) Disable SSL client certificate authentication.
   b. Configure S1 for outgoing connections.
      1) Disable identity assertion.
      2) Enable user ID and password authentication.
      3) Disable SSL.

3. Configure the S2 server.

   In the administrative console, the S2 server is configured for incoming requests to support identity assertion and to accept SSL connections. Configuration for outgoing requests and connections are not relevant for this scenario.

   a. Disable identity assertion.
   b. Enable user ID and password authentication.
   c. Disable SSL.

Authentication protocol for EJB security

WebSphere Application Server Version 8.5 servers support the CSIv2 authentication protocol only. SAS is only supported between Version 6.0.x and earlier version servers that have been federated in a Version 8.5 cell. The option to select between SAS, CSIv2, or both is only available in the administration console when a Version 6.0.x or earlier release has been federated in a Version 8.5 cell.

SAS is the authentication protocol used by all previous releases of WebSphere Application Server and is maintained for backwards compatibility. The Object Management Group (OMG) has defined the authentication protocol called CSIv2 so that vendors can interoperate securely. CSIv2 is implemented in WebSphere Application Server with more features than SAS and is considered the strategic protocol.

Important: SAS is supported only between Version 6.0.x and previous version servers that have been federated in a Version 6.1 cell.

Invoking Enterprise Java Beans (EJB) methods in a secure WebSphere Application Server environment requires an authentication protocol to determine the level of security and the type of authentication that occur between any given client and server for each request. It is the job of the authentication protocol during a method invocation to merge the server authentication requirements that are determined by the object Interoperable Object Reference (IOR) with the client authentication requirements that are determined by the client configuration and come up with an authentication policy specific to that client and server pair.

The authentication policy makes the following decisions, among others, which are all based on the client and server configurations:

- What kind of connection can you make to this server--Secure Sockets Layer (SSL) or TCP/IP?
- If SSL is chosen, how strong is the encryption of the data?
- If SSL is chosen, do you authenticate the client using client certificates?

- Do you authenticate the client with a user ID and password? Does an existing credential exist?
- Do you assert the client identity to downstream servers?
- Given the configuration of the client and server, can a secure request proceed?

You can configure both protocols (SAS and CSIv2) to work simultaneously. If a server supports both protocols, it exports an IOR containing tagged components describing the configuration for SAS and CSIv2. If a client supports both protocols, it reads tagged components for both CSIv2 and SAS. If the client supports both and the server supports both, CSIv2 is used. However, if the server supports SAS (for example, it is a previous WebSphere Application Server release) and the client supports both, the client chooses SAS for this request because the SAS protocol is what both have in common.

Choose a protocol by specifying the `com.ibm.CSI.protocol` property on the client side and configuring through the administrative console on the server side. More details are included in the SAS and CSIv2 properties articles.

Common Secure Interoperability Specification, Version 2

The Common Secure Interoperability Specification, Version 2 (CSIv2) defines the Security Attribute Service (SAS) that enables interoperable authentication, delegation, and privileges. The CSIv2 SAS and SAS protocols are entirely different. The CSIv2 SAS is a subcomponent of CSIv2 that supports SSL and interoperability with the EJB Specification, Version 2.1.

Security Attribute Service

The Common Secure Interoperability Specification, Version 2 Security Attribute Service (CSIv2 SAS) protocol is designed to exchange its protocol elements in the service context of a General Inter-ORB Protocol (GIOP) request and reply messages that are communicated over a connection-based transport. The protocol is intended for use in environments where transport layer security, such as that available through Secure Sockets Layer (SSL) and Transport Layer Security (TLS), is used to provide message protection (that is, integrity and or confidentiality) and server-to-client authentication. The protocol provides client authentication, delegation, and privilege functionality that might be applied to overcome corresponding deficiencies in an underlying transport. The CSIv2 SAS protocol facilitates interoperability by serving as the higher-level protocol under which secure transports can be unified.

Connection and request interceptors

The authentication protocols that are used by WebSphere Application Server are add-on Interoperable Inter-ORB Protocol (IIOP) services. IIOP is a request-and-reply communications protocol that is used to send messages between two Object Request Brokers (ORBs). For each request made by a client ORB to a server ORB, an associated reply is made by the server ORB back to the client ORB. Prior to any request flowing, a connection between the client ORB and the server ORB must be established over the TCP/IP transport (SSL is a secure version of TCP/IP). The client ORB invokes the authentication protocol client connection interceptor, which is used to read the tagged components in the IOR of the object that is located on the server. As mentioned previously, the authentication policy is established here for the request. Given the authentication policy (a coalescing of the server configuration with the client configuration), the strength of the connection is returned to the ORB. The ORB makes the appropriate connection, usually over SSL.

After the connection is established, the client ORB invokes the authentication protocol client request interceptor, which is used to send security information other than what is established by the transport. The security information includes the user ID and password token that are authenticated by the server, an authentication mechanism-specific token that is validated by the server, or an identity assertion token. Identity assertion is a way for one server to trust another server without the need to re-authenticate or re-validate the originating client. However, some work is required for the server to trust the upstream server. This additional security information is sent with the message in a service context. A service context has a registered identifier so that the server ORB can identify which protocol is sending the information.

The fact that a service context contains a unique identity is another way for WebSphere Application Server to support both SAS and CSIv2 simultaneously because both protocols have different service context IDs. After the client request interceptor finishes adding the service context to the message, the message is sent to the server ORB.

When the message is received by the server ORB, the ORB invokes the authentication protocol server request interceptor. This interceptor looks for the service context ID known by the protocol. When both SAS and CSIv2 are supported by a server, two different server request interceptors are invoked and both interceptors look for different service context IDs.

However, only one finds a service context for any given request. When the server request interceptor finds a service context, it reads the information in the service context. A method is invoked to the security server to authenticate or validate client identity. The security server either rejects the information or returns a credential. A credential contains additional information about the client that is retrieved from the user registry so that authorization can make the appropriate decision. Authorization is the process of determining if the user can invoke the request based on the roles that are applied to the method and the roles given to the user.

If a service context is not found by the CSIv2 server request interceptor, the interceptor process looks at the transport connection to see if a client certificate chain is sent. This process is done when SSL client authentication is configured between the client and server.

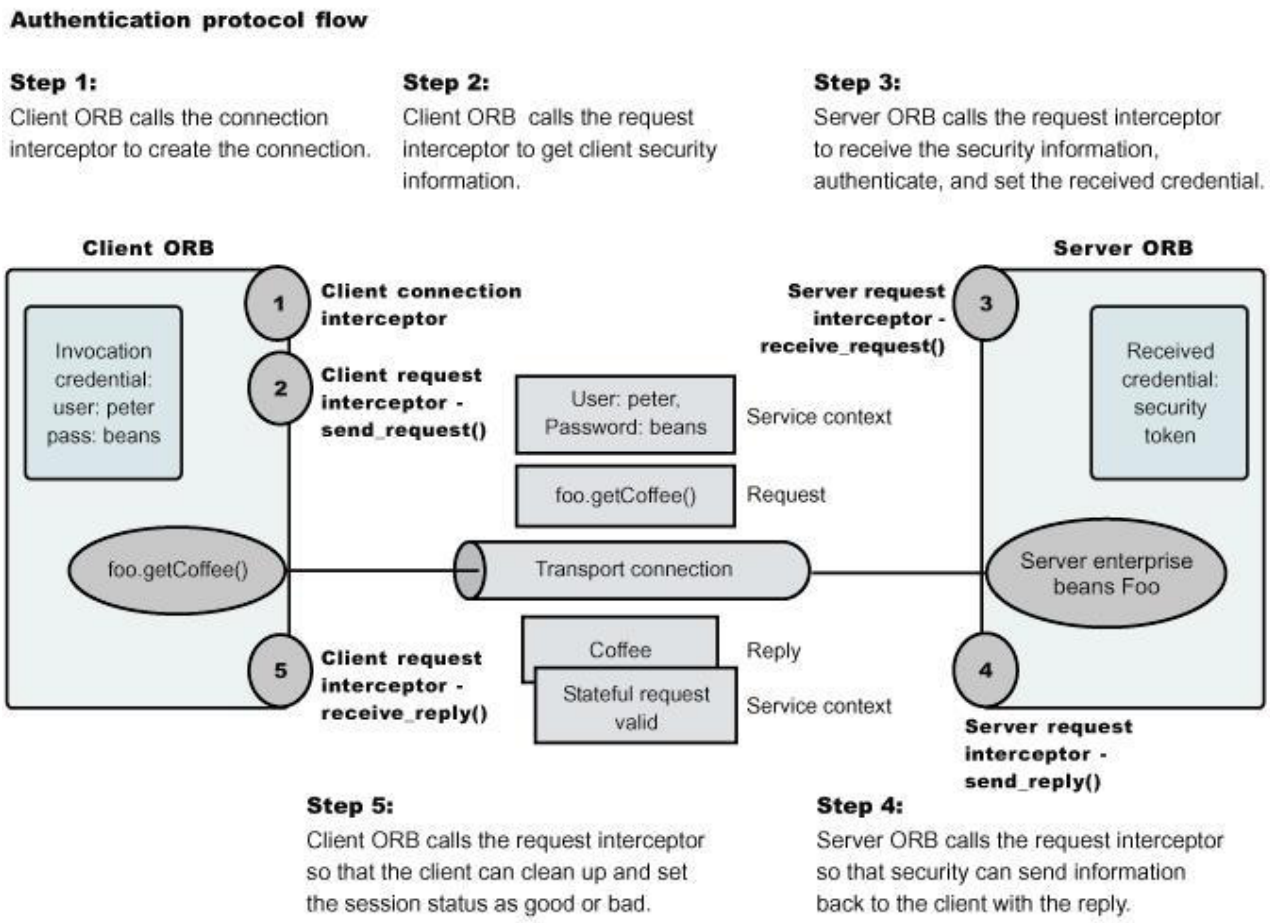
If a client certificate chain is found, the distinguished name (DN) is extracted from the certificate and is used to map to an identity in the user registry. If the user registry is Lightweight Directory Access Protocol (LDAP), the search filters defined in the LDAP registry configuration determine how the certificate maps to an entry in the registry. If the user registry is local OS, the first attribute of the distinguished name (DN) maps to the user ID of the registry. This attribute is typically the common name.

If the certificate does not map, no credential is created and the request is rejected. When valid security information is not presented, the method request is rejected and a NO_PERMISSION exception is sent back with the reply. However, when no security information is presented, an unauthenticated credential is created for the request and the authorization engine determines if the method gets invoked. For an unauthenticated credential to invoke an Enterprise JavaBeans (EJB) method, either no security roles are defined for the method or a special Everyone role is defined for the method.

When the method invocation is completed in the EJB container, the server request interceptor is invoked again to complete server authentication and a new reply service context is created to inform the client request interceptor of the outcome. This process is typically for making the request stateful. When a stateful request is made, only the first request between a client and server requires that security information is sent. All subsequent method requests need to send a unique context ID only so that the server can look up the credential that is stored in a session table. The context ID is unique within the connection between a client and server.

Finally, the method request cycle is completed by the client request interceptor receiving a reply from the server with a reply service context providing information so that the client-side stateful context ID can be confirmed and reused.

Specifying a stateful client is done through the property com.ibm.CSI.performStateful (true/false). Specifying a stateful server is done through the administrative console configuration.



. *Authentication protocol flow*

Authentication policy for each request

The authentication policy of a given request determines the security protection between a client and a server. A client or server authentication protocol configuration can describe required features, supported features, and non-supported features. When a client requires a feature, it can talk only to servers that either require or support that feature. When a server requires a feature, it can talk only to clients that either require or support that feature. When a client supports a feature, it can talk to a server that supports or requires that feature, but can also talk to servers that do not support the feature. When a server supports a feature, it can talk to a client that supports or requires the feature, but can also talk to clients that do not support the feature or chose not to support the feature.

For example, for a client to support client certificate authentication, some setup is required to either generate a self-signed certificate or to get one from a certificate authority (CA). Some clients might not need to complete these actions, therefore, you can configure this feature as not supported. By making this decision, the client cannot communicate with a secure server that requires client certificate authentication. Instead, this client can choose to use the user ID and password as the method of authenticating itself to the server.

Typically, supporting a feature is the most common way of configuring features. It is also the most successful during runtime because it is more forgiving than requiring a feature. Knowing how secure servers are configured in your domain, you can choose the right combination for the client to ensure successful method invocations and still get the most security. If you know that all of your servers support both client certificate and user ID and password authentication for the client, you might want to require one

and not support the other. If both the user ID and password and the client certificate are supported on the client and server, both are performed, but user ID and password take precedence at the server. This action is based on the CSIv2 specification requirements.

Authentication protocol support

Use this page to reference information regarding supported authentication protocols.

Authentication protocol support

Beginning with WebSphere Application Server Version 8.5, the WebSphere Application Server Version 8.5 servers only support the Common Secure Interoperability Version 2 (CSIv2) authentication protocol. Secure Authentication Service (SAS) is only supported between Version 6.0.*x* and previous version servers that have been federated in a Version 8.5 cell. The option to select between SAS, CSIv2, or both will only be made available in the administration console when a Version 6.0.x or previous release has been federated in a Version 8.5 cell.

In future releases, IBM will no longer ship or support the Secure Authentication Service (SAS) IIOP security protocol. It is recommended that you use the Common Secure Interoperability version 2 (CSIv2) protocol.

You can configure both protocols to work simultaneously between Version 6.0.*x* and previous version servers that have been federated in a Version 8.5 cell. If a server supports both protocols, it exports an interoperable object reference (IOR) that contains tagged components describing the configuration for SAS and CSIv2. If a client supports both protocols, it reads tagged components for both CSIv2 and SAS. If the client and server support both protocols, CSIv2 is used. However, if the server supports SAS (for example, the server is a previous WebSphere Application Server release) and the client supports both protocols, the client chooses SAS for this request.

Choose a protocol using the com.ibm.CSI.protocol property on the client side and configure this protocol through the administrative console on the server side.

You can configure both protocols to work simultaneously. If a server supports both protocols, it exports an interoperable object reference (IOR) that contains tagged components describing the configuration for SAS and CSIv2. If a client supports both protocols, it reads tagged components for both CSIv2 and SAS. If the client and the server support both protocols, CSIv2 is used. However, if the server supports SAS (for example, it is a previous WebSphere Application Server release) and the client supports both protocols, the client chooses SAS for this request.

Common Secure Interoperability Version 2 features

The following Common Secure Interoperability Version 2 (CSIv2) features are available in IBM WebSphere Application Server: message layer authentication, identity assertion, and security attribute propagation.

- Identity Assertion

  Supports a downstream server in accepting the client identity that is established on an upstream server, without having to authenticate again. The downstream server trusts the upstream server.
- Message Layer Authentication

  Authenticates credential information and sends that information across the network so that a receiving server can interpret it.
- Security attribute propagation

  Supports the use of the authorization token to propagate serialized Subject contents and PropagationToken contents with the request. You can propagate these objects using a pure client or a server login that adds custom objects to the Subject. Propagating security attributes prevents downstream logins from having to make user registry calls to look up these attributes.

Propagating security attributes is also useful when the security attributes contain information that is only available at the time of authentication. This information cannot be located using the user registry on downstream servers.

Identity assertion to the downstream server

When a client authenticates to a server, the received credential is set. When the authorization engine checks the credential to determine whether access is permitted, it also sets the *invocation* credential . *Identity assertion* is the invocation credential that is asserted to the downstream server.

When a client authenticates to a server, the received credential is set. When the authorization engine checks the credential to determine whether access is permitted, it also sets the *invocation* credential so that if the Enterprise JavaBeans (EJB) method calls another EJB method that is located on other servers, the invocation credential can be the identity used to invoke the downstream method. Depending on the RunAs mode for the enterprise beans, the invocation credential is set as the originating client identity, the server identity, or a specified different identity. Regardless of the identity that is set, when identity assertion is enabled, it is the invocation credential that is asserted to the downstream server.

The invocation credential identity is sent to the downstream server in an identity token. In addition, the sending server identity, including the password or token, is sent in the client authentication token when basic authentication is enabled. The sending server identity is sent through a Secure Sockets Layer (SSL) client certification authentication when client certificate authentication is enabled. Basic authentication takes precedence over client certificate authentication.

Both identity tokens are needed by the receiving server to accept the asserted identity. The receiving server completes the following actions to accept the asserted identity:

- The server determines whether the sending server identity, sent with a basic authentication token or with an SSL client certificate, is on the trusted principal list of the receiving server. The server determines whether the sending server can send an identity token to the receiving server.
- After it is determined that the sending server is on the trusted list, the server authenticates the sending server to verify its identity.
- The server is authenticated by comparing the user ID and password from the sending server to the receiving server. If the credentials of the sending server are authenticated and on the trusted principal list, then the server proceeds to evaluate the identity token.
- The downstream server checks its defined user registry for the presence of the asserted user ID to gather additional credential information for authorization purposes (for example, group memberships). Thus, the downstream user registry must contain all of the asserted user IDs. Otherwise, identity assertion is not possible. In a stateful server, this action occurs once for the sending server and the receiving server pair where the identity tokens are the same. Subsequent requests are made through a session ID.

  Note: When the downstream server does not have a user registry with access to the asserted user IDs in its repository, do not use identity assertion because authorization checks will fail. By disabling identity assertion, the authorization checks on the downstream server are not needed.

Evaluation of the identity token consists of the following four identity formats that exist in an identity token:

- Principal name
- Distinguished name
- Certificate chain
- Anonymous identity

The product servers that receive authentication information typically support all four identity types. The sending server decides which one is chosen, based on how the original client authenticated. The existing type depends on how the client originally authenticates to the sending server. For example, if the client uses Secure Sockets Layer (SSL) client authentication to authenticate to the sending server, then the

identity token sent to the downstream server contains the certificate chain. With this information, the receiving server can perform its own certificate chain mapping and interoperability is increased with other vendors and platforms.

After the identity format is understood and parsed, the identity maps to a credential. For an ITTPrincipal identity token, this identity maps one-to-one with the user ID fields.

For an ITTDistinguishedName identity token, the mapping depends on the user registry. For Lightweight Directory Access Protocol (LDAP), the configured search filter determines how the mapping occurs. For LocalOS, the first attribute of the distinguished name (DN), which is typically the same as the common name, maps to the user ID of the registry.

Identity assertion is only available using the Common Secure Interoperability Version 2 (CSIv2) protocol.

Note: There is a restriction for using identity assertion with KRB token to downstream. If you use identity assertion with Kerberos enabled, the identity assertion does not have the Kerberos authentication token (KRBAuthnToken) when going to downstream servers. It uses LTPA for authentication instead.

Identity assertions with trust validation

If you want an application or system provider to perform an identity assertion with trust validation, it can be accomplished by use of the Java Authentication and Authorization Service (JAAS) login framework, where trust validation is performed in one login module and credential creation in another. These two custom login modules are used to create a JAAS login configuration that performs a login to an identity assertion.

Two custom login module are required:

- A user-implemented trust association login module. This login module performs whatever trust verification the user requires. When trust is verified, the trust verification status and the login identity must be placed in a map in the share state of the login module to enable the credential creation login module to use that information. The map must be stored in the com.ibm.wsspi.security.common.auth.module.IdentityAssertionLoginModule.state property. State maps contain the following information:
  - com.ibm.wsspi.security.common.auth.module.IdentityAssertionLoginModule.trusted – set to `true`, if trusted, and `false`, if not trusted.
  - com.ibm.wsspi.security.common.auth.module.IdenityAssertionLoginModule.principal – contains the principal of the identity.
  - com.ibm.wsspi.security.common.auth.module.IdentityAssertionLoginModule.certificates – contains the certificate of the identity
- The com.ibm.wsspi.security.common.auth.module.IdentityAssertionLoginModule module performs the credential creation. It requires that the trust state information be in the login context's shared state. This login module is protected by the Java 2 security runtime permissions for the following:
  - com.ibm.wsspi.security.common.auth.module.IdentityAssertionLoginModule.initialize
  - com.ibm.wsspi.security.common.auth.module.IdentityAssertionLoginModule.login

IdentityAssertionLoginModule searches for the trust information in the shared state property, com.ibm.wsspi.security.common.auth.module.IdentityAssertionLoginModule.state. This is a map that contains the trust status and the identity used to login. The map includes the following:

- com.ibm.wsspi.security.common.auth.module.IdentityAssertionLoginModule.trusted – if set to `true` it is trusted, `false` if not trusted.
- com.ibm.wsspi.security.common.auth.module.IdentityAssertionLoginModule.principal – if a principal is used, it contains the principal of the identity necessary to login.
- com.ibm.wsspi.security.common.auth.module.IdentityAssertionLoginModule.certificates – if a certificate is used, it contains an array of a certificate chain that includes the identity necessary to login.

A WSLoginFailedException is returned if the state, trust, or identity information is missing. The login module then performs a login of the identity. The subject now contains the new identity.

Message layer authentication

Defines the credential information and sends that information across the network so that a receiving server can interpret it.

When you send authentication information across the network using a token the transmission is considered message layer authentication because the data is sent with the message inside a service context.

A pure Java client uses Kerberos (KRB5) or basic authentication, or Generic Security Services Username Password (GSSUP), as the authentication mechanism to establish client identity.

However, a servlet can use either basic authentication (GSSUP) or the authentication mechanism of the server, Kerberos (KRB5) or Lightweight Third Party Authentication (LTPA), to send security information in the message layer. Use KRB5 or LTPA by authenticating or by mapping the basic authentication credentials to the security mechanism of the server.

The security token that is contained in a token-based credential is authentication mechanism-specific. The way that the token is interpreted is only known by the authentication mechanism. Therefore, each authentication mechanism has an object ID (OID) representing it. The OID and the client token are sent to the server, so that the server knows which mechanism to use when reading and validating the token. The following list contains the OIDs for each mechanism:

BasicAuth (GSSUP): oid:2.23.130.1.1.1
KRB5: OID: 1.2.840.113554.1.2.2
LTPA: oid:1.3.18.0.2.30.2
SWAM: No OID because it is not forwardable

Note: SWAM is deprecated in WebSphere Application Server Version 8.5 and will be removed in a future release.

On the server, the authentication mechanisms can interpret the token and create a credential, or they can authenticate basic authentication data from the client, and create a credential. Either way, the created credential is the `received` credential that the authorization check uses to determine if the user has access to invoke the method. You can specify the authentication mechanism by using the following property on the client side:

- com.ibm.CORBA.authenticationTarget

Basic authentication (BasicAuth) and KRB5 are currently the only valid values. You can configure the server through the administrative console.

Note: When perform basic authentication is enabled, if the client is not similarly configured (and does not pass a credential such as a user ID and password).

Configuring authentication retries

Situations occur where you want a prompt to display again if you entered your user ID and password incorrectly or you want a method to retry when a particular error occurs back at the client. If you can correct the error by information at the client side, the system automatically performs a retry without the client seeing the failure, if the system is configured appropriately.

Some of these errors include:
- Entering a user ID and password that are not valid
- Having an expired credential on the server

- Failing to find the stateful session on the server

By default, authentication retries are enabled and perform three retries before returning the error to the client. Use the com.ibm.CORBA.authenticationRetryEnabled property (`True` or `False`) to enable or disable authentication retries. Use the com.ibm.CORBA.authenticationRetryCount property to specify the number of retry attempts.

Immediate validating of a basic authentication login

In WebSphere Application Server Version 6.x, a behavior is defined during request_login for a BasicAuth login. In releases prior to Version 5, a BasicAuth login takes the user ID and password entered through the loginSource method and creates a BasicAuth credential. If either the user ID or the password is not valid, the client program does not find out until the first method request is attempted. When the user ID or password is specified during a prompt or programmatic login, the user ID and password are authenticated by default with the security server, with a `True` or `False` returned as the result. If `False`, an `org.omg.SecurityLevel2.LoginFailed` exception is returned to the client indicating that the user ID and password are not valid. If `True`, then the BasicAuth credential is returned to the caller of the request_login. To disable this feature on the pure client, specify `com.ibm.CORBA.validateBasicAuth=false`. By default, this feature is set to `True`. On the server side, specify this property in the security dynamic properties.

Using Microsoft Active Directory for authentication

WebSphere Application Server supports the Microsoft Active Directory. Many installations use the Microsoft Active Directory as their primary component for managing user authentication and user data. Authenticating a user across multiple repositories or across a distributed Lightweight Directory Access Protocol (LDAP), such as a Microsoft Active Directory forest can be challenging. In any search of the whole registry, if there is more than one match at run time, authentication fails because ambiguous matches result.

About this task

User IDs are guaranteed to be unique within a single domain, but there is no automatic guarantee that a given user ID is unique across a tree or a forest. The following figure exemplifies the condition of a given user ID not being unique across a tree or forest.

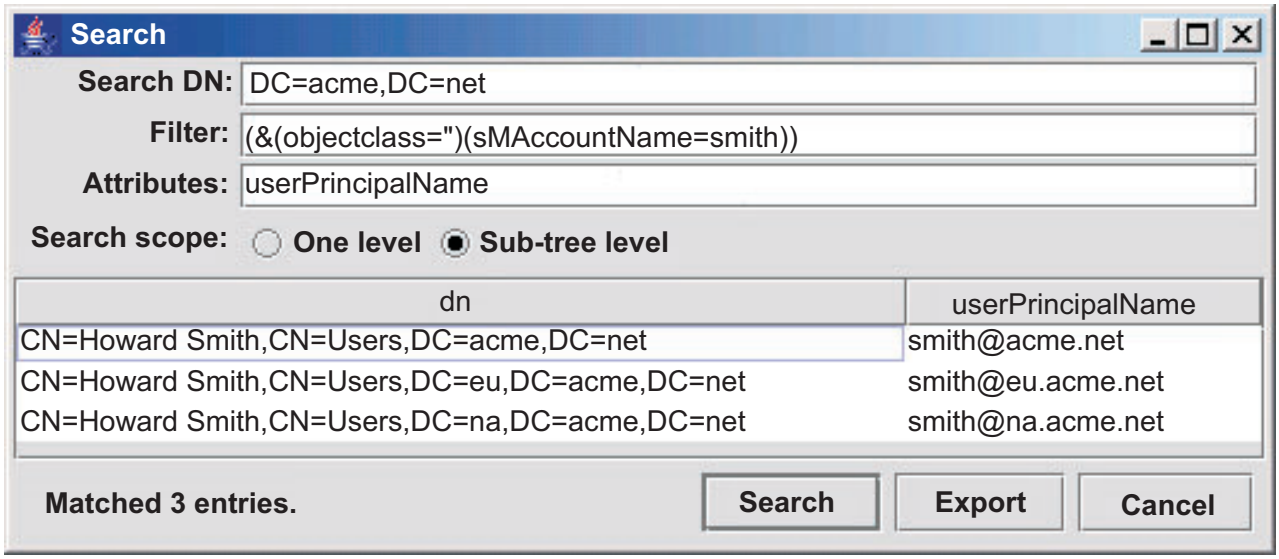


Figure 23. Forest search strategy.. Search illustration of a non-unique sAMAccountName across the entire forest.

Authenticating users across trees or forests can be a difficult task and the following steps should be performed.

Note: Windows You must ensure that the Microsoft Windows Computer Browser Service is enabled in your operating system when the following conditions are true:

- Your primary domain is managed by Microsoft Active Directory.
- The Primary Domain Controller (PDC) exists in a different subnet from WebSphere Application Server.
- You set the user registry for WebSphere Application Server to local OS and not Lightweight Directory Access Protocol (LDAP).

For more information on how to set and verify that the Microsoft Windows Computer Browser Service is enabled, see the Microsoft documentation for your operating system.

Procedure

1. Analyze the Microsoft Active Directory construct that defines your installation. Your analysis can conclude with the following forms:
   - Single LDAP registry - Simple configuration.
   - Federated repository (a forest)- Typical configuration.
   - Merger of federated repositories (a merger of trees into a forest)- Less typical configuration
   - Combination of user and group forests - Rare configuration
2. Develop strategies for user look up that match your Microsoft Active Directory installation. Remember that user IDs are guaranteed to be unique within a single domain, but there is no automatic guarantee that a given user ID is unique across a tree or a forest.
3. Evaluate with testing to ensure that your authentication search strategies successfully authenticate users in your Microsoft Active Directory installation.

Results

You will be in the position to authenticate users with LDAP registries in a Microsoft Active Directory forest.

What to do next

gotcha: When you select any of these scenarios, consult appropriate Microsoft Active Directory information to completely understand any implications the scenarios might have on your configuation planning.

Authentication using Microsoft Active Directory

Many installations use the Microsoft Active Directory as their primary component for managing user authentication and user data. One portion of the Microsoft Active Directory provides a Lightweight Directory Access Protocol (LDAP) service. WebSphere Application Server supports LDAP and, therefore, WebSphere Application Server supports the Microsoft Active Directory.

While the Microsoft Active Directory is fully LDAP-compliant, it exposes LDAP information in ways that can make it difficult to obtain directory information for WebSphere Application Server.

WebSphere Application Server operates in a way that assumes that a single LDAP directory contains all the information necessary to operate. With complex Microsoft Active Directory configurations, this is not the case. WebSphere Application Server - Microsoft Active Directory installations must handle unique challenges because of the way data is spread throughout the domain controllers in a forest.

Microsoft Active Directory installations frequently incorporate the use of a forest. As such, security questions pertaining to user ID uniqueness, reliably obtaining user group information, and group membership spread across forests become important.

The following figure highlights a typical Microsoft Active Directory installation environment.

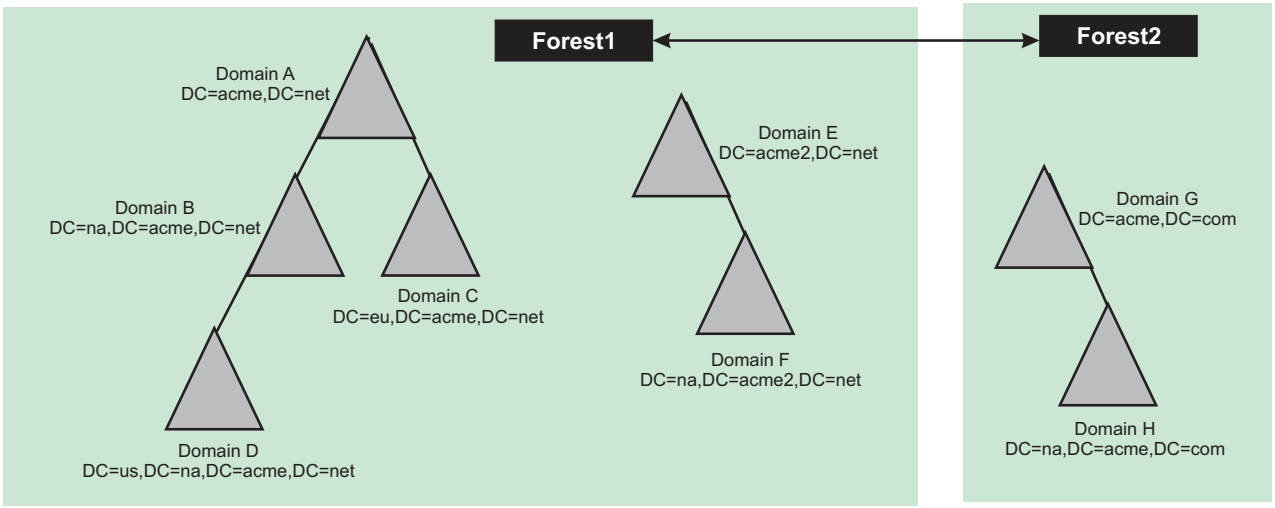


Figure 24. Microsoft Active Directory forests. An illustration of Microsoft Active Directory forests.

This figure illustrates two forests of one or more trees. A tree can contain one or more domains where the domain is the single atomic unit that forms the basis for the constructed environment. Each domain is made up of the primary domain components of the distinguished name (DN), for example, `dc=acme`, `dc=com`. A forest can extend trust to other forests (This trust is based on Kerberos.).

Microsoft Active Directory configurations with WebSphere Application Server

There can be a variety of Microsoft Active Directory configurations for WebSphere Application Server, which include:

- Simple configuration
- Typical configuration
- Less typical configurations
- Rare configurations

The simplest configuration consists of a stand-alone LDAP registry representing a single domain. This configuration represents the closest fit between WebSphere Application Server and the Microsoft Active Directory. In this configuration, Microsoft Active Directory is supported through the WebSphere Application Server stand-alone LDAP user registry implementation. Alternatively, you can access this single Microsoft Active Directory domain through a federated repositories registry, which contains a single LDAP repository.

Beyond the simple single domain Microsoft Active Directory configuration, a typical Microsoft Active Directory configuration consists of a single tree in a forest where each branch of the tree is a domain. An example of this configuration, which consists of a single tree of four domains (A, B, C, D), is shown in the following example:

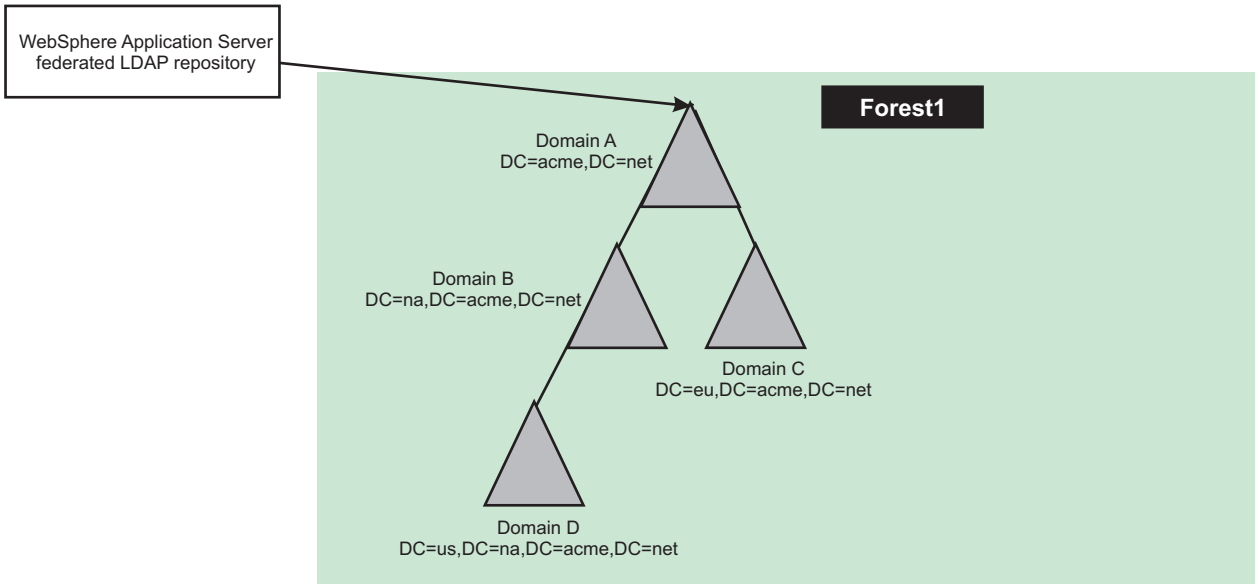


Figure 25. Typical forest configuration. Typical forest configuration.

Configurations, such as this configuration, frequently have domains that are organized by geography or organizational unit. The WebSphere Application Server registry configuration that is necessary to use this “single tree”Microsoft Active Directory implementation needs to use the federated repositories. This configuration contains an LDAP registry to map entries from multiple individual user repositories into a single virtual repository. These configurations create a federated user repository with a single named realm and an LDAP subtree within the single repository. The root of the repository is mapped to a base entry within the federated repository, which is the starting point within the hierarchical namespace of the virtual realm. LDAP searches in this configuration proceed with binding to the top domain object and following LDAP referrals.

gotcha: The stand-alone LDAP registry in WebSphere Application Server does not support LDAP referrals and cannot be used in a WebSphere Application Server - Microsoft Active Directory configuration.

Less typical WebSphere Application Server - Microsoft Active Directory configurations evolve from mergers of organizations units in a larger enterprise. Where a single forest of domains once served the enterprise, the merger of several new organizational units can add trees to the forest or even add more than a single forest to the environment. In this environment, the WebSphere Application Server LDAP configuration requires more careful design. You must use the federated repositories registry in such an environment with separate LDAP repositories mapped to the top of each tree in the forest. Again, if a Microsoft Active Directory tree exists under the top-level domain, LDAP referrals must be enabled for the LDAP registry. The forest resulting from a merger can look like the following figure:

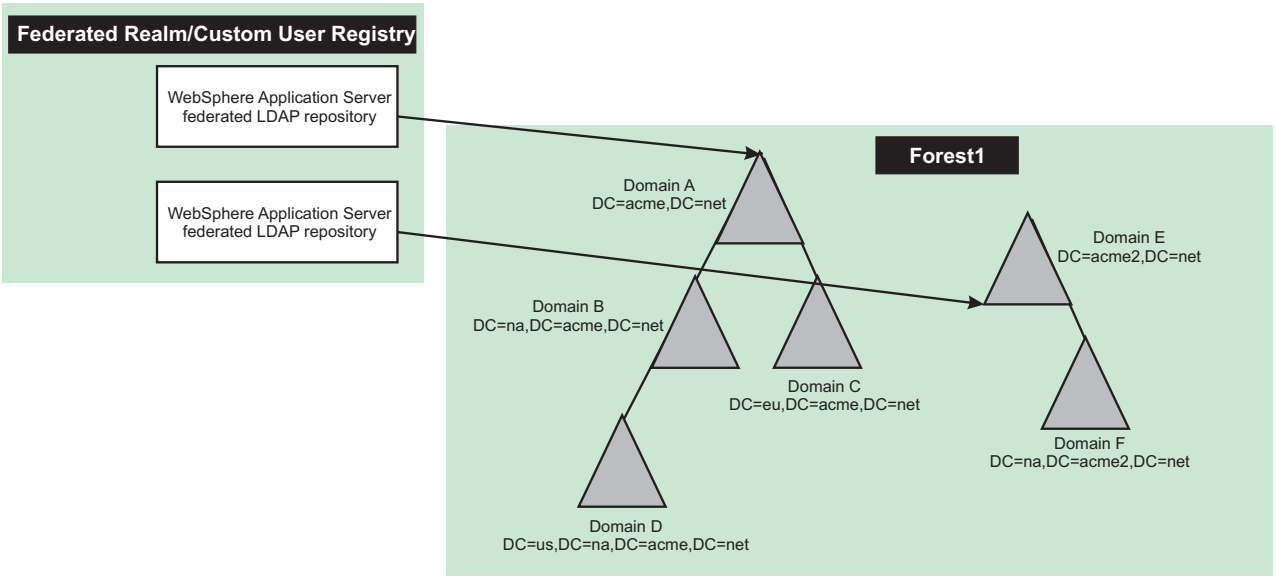


Figure 26. Less typical configurations. Less typical configurations that depict the merger of trees

Rare configurations consist of Microsoft Active Directory domains that are configured where there is a combination of a user forest and a group forest. Users are imported as ForeignSecurityPrincipals objects in the group forest. The groups contain the distinguished names (DN) of the ForeignSecurityPrincipals objects as members.

In this form of configuration, direct group lookups do not occur. Lookups are relegated to a static group query across multiple registries. This configuration requires a custom user registry. However, WebSphere Application Server registries do not support this type of configuration. See the following figure.

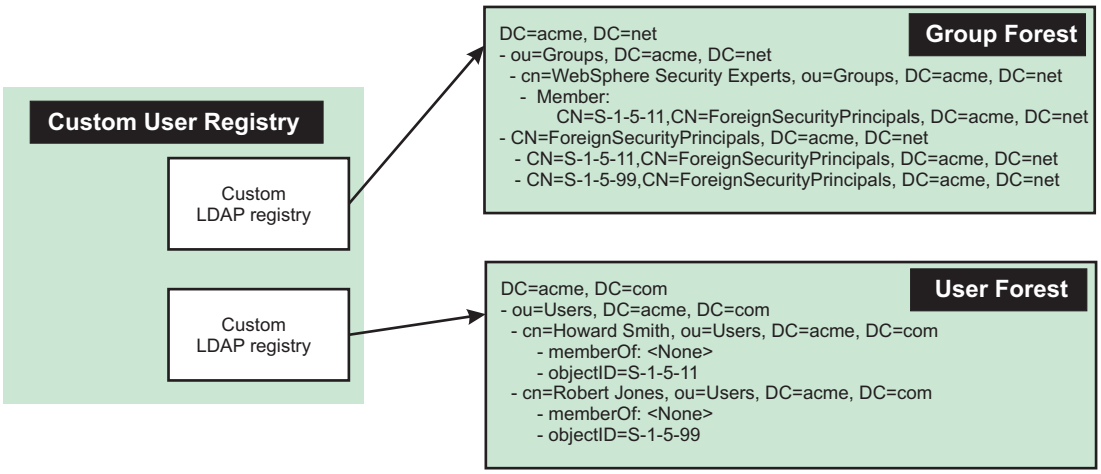


Figure 27. Resource model forest. An illustration of a resource model forest.

Using a Microsoft Active Directory forest as LDAP - user filter

Authenticating a user across multiple repositories, or across a distributed LDAP, such as a Microsoft Active Directory forest configuration can be challenging. In any search of the whole registry, authentication fails if there is more than one match at run time because ambiguous matches result. In multiple Microsoft Active Directory domain environment, the WebSphere Application Server administrator must consider that the default unique ID in the Microsoft Active Directory is the sAMAccountName attribute of a user. User IDs

are guaranteed to be unique within a single domain, but it is not possible to guarantee that a given user ID is unique across a tree or a forest. See the topic, “Authenticating users with LDAP registries in a Microsoft Active Directory forest”to understand how to search for user IDs within a Microsoft Active Directory forest using the sAMAccountName attribute of a user.

gotcha: Before selecting any of these scenarios, consult appropriate Microsoft Active Directory information to completely understand any implications the scenarios might have on your configuration planning.

Groups spanning domains with Microsoft Active Directory

The domains and forests functional levels of the Microsoft Active Directory control which configurations are available for use. How you configure Microsoft Active Directory affects how group membership is determined within WebSphere Application Server. Using groups to configure your Microsoft Active Directory installation with the product allows flexible management.

A breakdown follows of applicable functional levels that apply to a Microsoft Active Directory installation with the product.

- Domain Functional Levels
  - Native
    - Supported by Windows Server 2008 and Windows Server 2008 R2
    - Default in Windows 2008

  You must use native domain functional levels to support group nesting, and universal groups. Forest functional levels do not directly affect group membership. The Windows 2008 operating system is the exception.
- Forest Functional Levels
  - Windows Server 2008 or Windows Server 2008 R2
    - All domains operate at the Windows Server 2008 **domain functional level**.

      If the forest functional level is set to Windows Server 2008, then that also makes the domain functional level for all domains to be Windows Server 2008 Native level, which adds to the group nesting and Universal groups features to Microsoft Active Directory.

Microsoft Active Directory groups

In a domain, Microsoft Active Directory provides support for different types of groups and group scopes. Groups in Microsoft Active Directory are containers with other objects within them as members. Those objects can be user objects, other group objects, which is group nesting, and other objects types, such as computers. The group type determines the type of task that you manage with the group. The group scope determines whether the group can have members from multiple domains or a single domain. In summary:

- Groups are typically a collection of user accounts.
- Members receive permission given to groups.
- Users can be members of multiple groups.
- Groups can be members of other groups, which are nested groups.

gotcha: In WebSphere Application Server, security roles of the individual, which map to application permissions or authorizations, must be bound to either users or groups at application deployment time. From an administrative point of view, it is preferable to assign permissions once for a group instead of assigning permissions repeatedly for each user account. Then the ability to act in a given role is under the control of the directory administrator, instead of the WebSphere administrator. Because the job of the directory administrator is to create and delete users, change group memberships for users, and other tasks, this approach is generally the correct division of responsibilities.

Group types determine how the group is used. The Microsoft Active Directory group types are:

- **Security groups:** Microsoft Active Directory uses security groups for granting permissions to gain access to resources.
- **Distribution groups:** Distribution groups are used by Windows-based applications as lists for nonsecurity-related functions. Distribution groups are used for sending email messages to groups of users. You cannot grant Windows permissions to distribution groups.

Although WebSphere Application Server can use either type of group, security groups are typically bound to WebSphere Application Server security roles.

Group scopes describe which type of objects can be arranged together within a group. Group nesting describes when one group is a member of other groups. The Microsoft Active Directory group scopes are:

- **Domain local group:**
  - **Windows usage:** Members of this group can come from any domain, but can access Windows resources only in the local domain. Use this scope to grant permissions to domain resources that are located in the same domain in which you created the domain local group. Domain local groups can exist in all mixed, native, and interim functional level of domains and forests.
  - **Restriction:** You cannot define group nesting in a domain local group. A domain local group cannot be a member of another domain local group or any other group in the same domain.
  - **WebSphere usage:** Users are not typically placed in domain - local groups due to these restrictions. WebSphere Application Server security roles are not typically bound to domain local groups.
- **Global Group:**
  - **Windows usage:** Members of this group originate from a local domain, but can access Windows resources in any domain. The global group is used to organize users who share similar Windows network access requirements. You can add members only from the domain in which the global group is created. You can use this group to assign permissions to gain access to Windows resources that are located in any domain in the domain, tree, or forest.

    You can group users with similar function under global scope and give permission to access a Windows resource, such as a printer or shared folder and files, that is available in local or another domain in the same forest. You can use global groups to grant permission to gain access to Windows resources that are located in any domain in a single forest as their memberships are limited. You can add user accounts and global groups only from the domain in which global group is created.

    Nesting is possible for global groups within other groups as you can add a global group into another global group from any domain. Members of a global group can be members of a domain - local group. Global groups exist in all mixed, native, and interim functional levels of domains and forests.

  WebSphere Application Server usage: Global groups are visible on every domain controller, but memberships are only visible for local users. That is, you can see your group memberships only if you query your home domain controller. A global group should contain groups of users. Global groups are intended to be included in universal groups.
- **Universal Group:**
  - **Windows usage:** Members in this group can come from any domain and access Windows resources in multiple domains. Universal group memberships are not limited like global groups. All domain user accounts and groups can be members of a universal group.
  - **Restrictions:**
    - Universal groups are available when the domain is at a Windows mixed functional level.
    - It can be expensive to replicate this data across the forest. Group definitions and deletions are relatively rare compared to the equivalent user actions, and nested group membership changes are typically rare compared to memberships of users within groups,

      gotcha: Consult appropriate Microsoft Active Directory information concerning any implications of replicating data across forests.

– **WebSphere usage:**
  - Universal Groups and their memberships are visible on every domain controller in the forest.
  - Universal groups are also visible when using the Global Catalog. To be useful, all user objects must be directly in the universal group,

Universal group guidelines

1. Assign permissions to universal groups for Windows resources in any domain in the network.
2. Use universal groups only when their membership is static. Changes in membership can cause excessive network traffic between domain controllers. Membership of universal groups can be replicated to many domain controllers.
3. Add global groups from several domains to a universal group.
4. Assign permissions for access to a Windows resource to the universal group and for use by WebSphere Application Server group membership resolution across multiple domains.
5. Use a universal group in the same way as a domain local group to assign resource permissions.

gotcha: When you select any of these scenarios, consult the appropriate Microsoft Active Directory information to completely understand any implications the scenarios might have on your configuation planning.

Microsoft Active Directory Global Catalog

A Global Catalog is a Global Catalog Server. A Global Catalog holds a full set of attributes for the domain in which it resides and a subset of attributes for all objects in the Microsoft Active Directory Forest. The primary two functions of a Global Catalog within the Microsoft Active Directory are logon capability and Microsoft Active Directory queries.

A Global Catalog in a Microsoft Active Directory installation with the product is a single Lightweight Directory Access Protocol (LDAP) repository that contains a subset of user information from all the domains in the forest. This information includes user IDs, authentication information, and groups, but not all the group information.

You can use the Global Catalog on any domain controller in the forest, even in subdomains. The Global Catalog is a solution to the WebSphere Application Server limitation of a "single registry". There are limitations to the Global Catalog. Users from the local domain controller contain group "memberOf" information. Users from a foreign domain controller contain limited "memberOf" information because the global group information is not replicated to every domain controller.

Nested global groups in universal groups

This is a typical structure of group membership and consists of the following characteristics:

- Users are distributed across domain controllers in a forest containing multiple domain controllers.
- Users are defined in global groups within their own local domain controller.
- A universal group contains the global groups, which reflects a Java Platform Enterprise Edition (Java EE) role that maps to a set of users spread across multiple domain controllers.

The following figure illustrates nested global groups in universal groups.

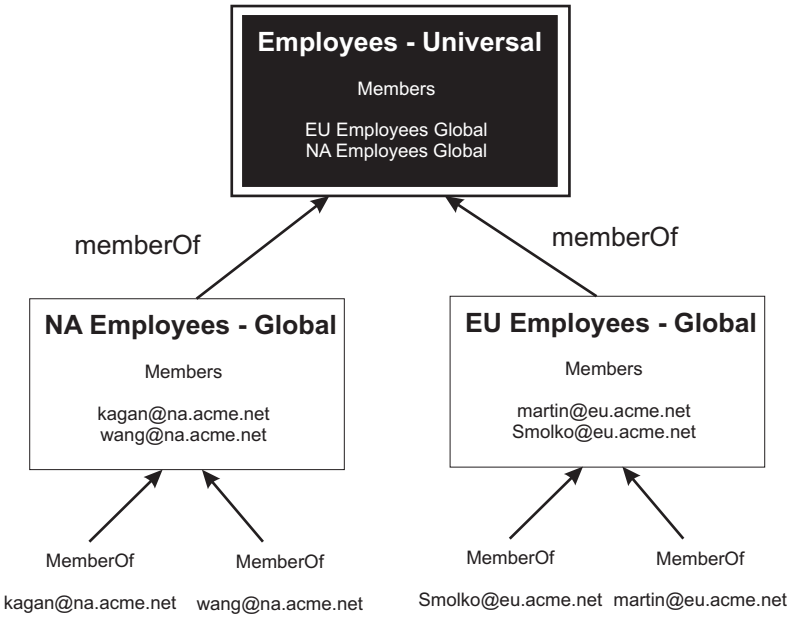


Figure 28. Nested global groups in universal groups. This figure illustrates nested global groups in universal groups.

It is a challenge to develop methods of configuring WebSphere Application Server to be able to find users and their group memberships when the information is spread across multiple domain controllers. One method requires that WebSphere Application Server follow LDAP referrals to find the home domain controller for each user and that WebSphere Application Server perform nested group queries.

gotcha: This approach does not use the Global Catalog.

Another method and the simplest approach has universal groups that contain users and uses a Global Catalog, which requires using referrals. The figure that follows illustrates this method.

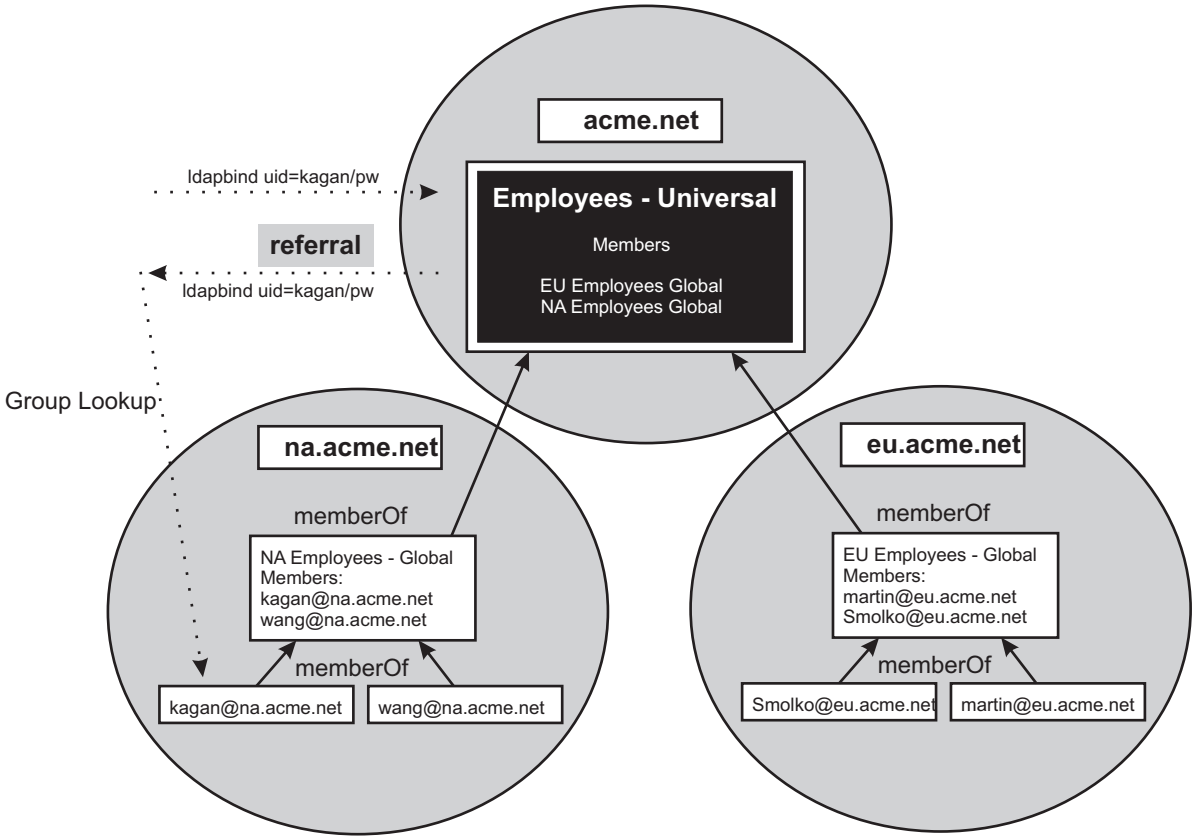


Figure 29. Locating group memberships. This figure illustrates the process of locating group memberships.

A variation on this method is to not use universal groups. You can use this approach when universal groups are not available.

gotcha: This approach does not use the Global Catalog.

You might consider using the Microsoft Active Directory Global Catalog as the WebSphere Application Server registry. There are three scenarios; however, the first two scenarios demonstrate how failures occur.

1. If you configure WebSphere Application Server to use Global Catalog as its LDAP registry and follow referrals, then individual users are visible in each domain controller. Because a user must exist only once in the registry, all logins fail.
2. If you configure WebSphere Application Server to use Global Catalog as its LDAP registry and do not follow referrals and the individual users are within global groups, then group membership is incomplete. See the following figure, which illustrates this limitation.

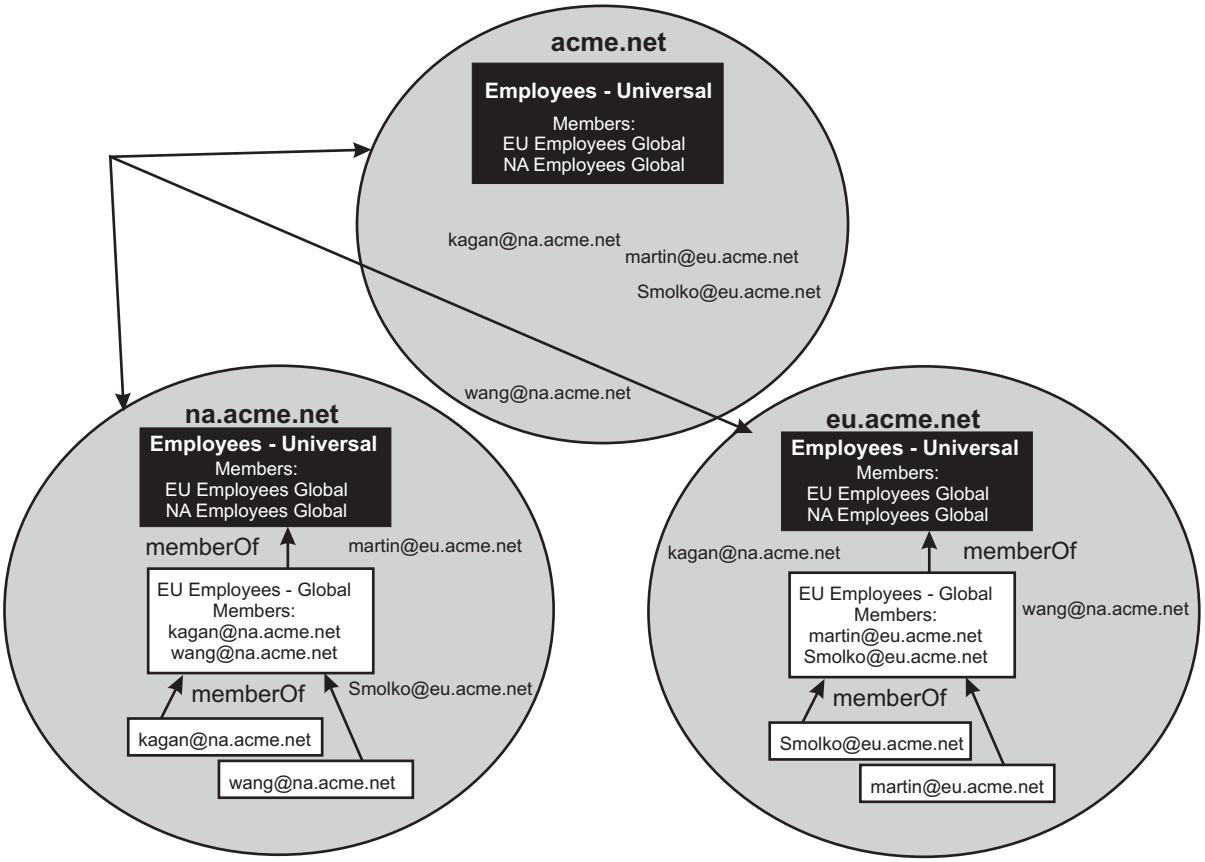


Figure 30. Global catalog (without using referrals). An illustration of a Global Catalog without using referrals

3. When you configure WebSphere Application Server to use Global Catalog as its LDAP registry, do not follow referrals, and users are directly contained within universal global groups, then group membership is complete.

gotcha: When you select any of these scenarios, consult appropriate Microsoft Active Directory information to completely understand any implications the scenarios might have on your configuation planning.

Options for finding group membership within a Microsoft Active Directory forest

Locating and finding group membership with the Microsoft Active Directory forest is necessary for authenticating users. There are several ways to approach finding group membership within the Microsoft Active Directory forest.

The following figure depicts an example of group membership with the Microsoft Active Directory forest. This figure is used to explain ways to find group membership.

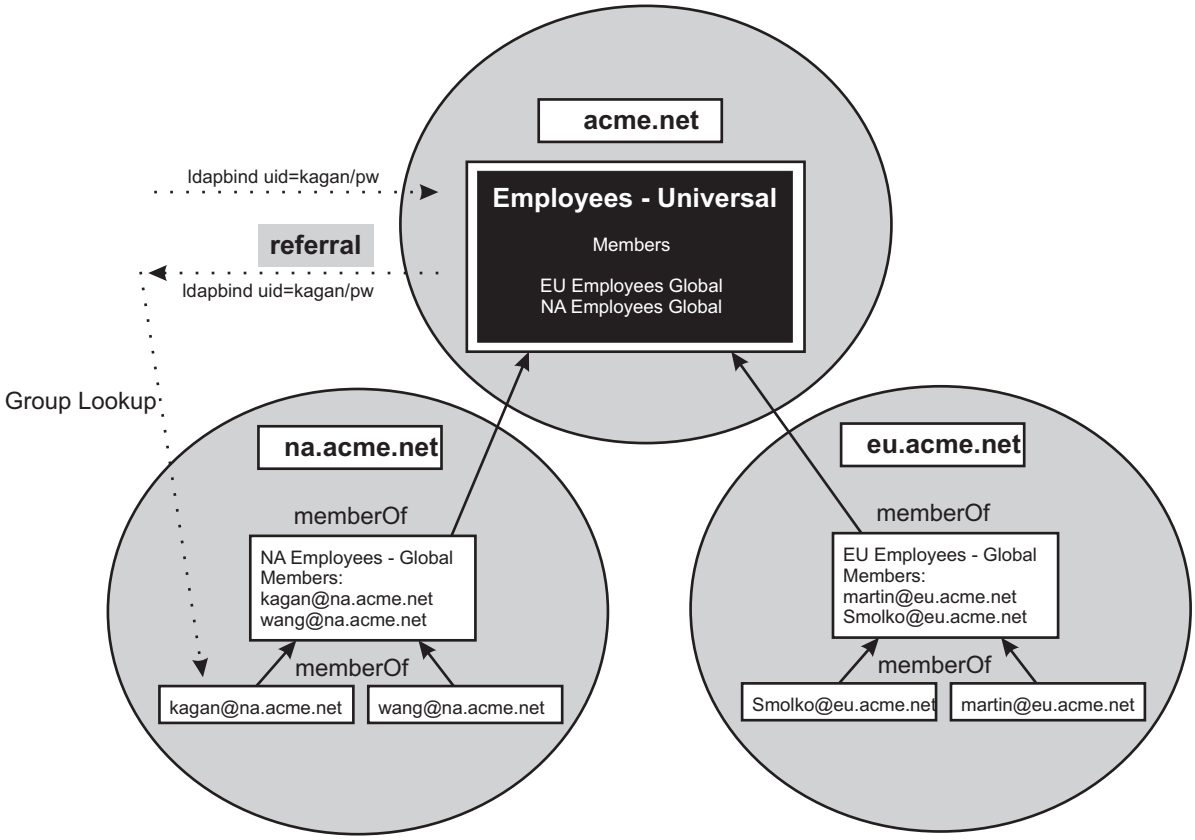


Figure 31. Finding group membership.. An illustration of ways to find group membership.

- **Option 1** does not use nested groups, and the following steps describe the process of locating group membership using a hypothetical organizational structure.
  - Create a global group of **NA employees**.
  - Create a global group of **EU employees**.
  - Map the Java Platform Enterprise Edition (Java EE) role to **NA employees** + **EU employees**. This mapping can become unmanageable if there are too many sub domains
  - Enable referrals.

    In WebSphere Application Server Version 6.1, use federated repositories, specifically:
    - Use a federated realm.
    - Add the Microsoft Active Directory top-level domain controller to the repository. Do not add sub-domain controllers. Doing this results in multiple matches when searches for user IDs occur. The multiple matches cause user logins to fail.
    - Select "**`Support referrals to other LDAP servers`**" = "`follow`".
- **Option 2** uses universal groups.
  - Put individual users into the universal group, **Employees**.

    Requirements:
    - The Windows 2003 Native domain functional levels is required.
    - Userids must be directly contained within universal groups.
  - Map Java EE role to **Employees**.
  - Connect to any global catalog in the forest.

Tip: This option reduces the amount of directory lookup traffic. WebSphere Application Server does not have to follow all the referrals across the directory tree. That is, each domain controller can fully resolve the group information locally.

- **Option 3** uses nested groups.
  - Create the universal group, **Employees**.
  - Create **NA Employees** and **EU Employees** as global groups and make them members in the **Employees** universal group.

    Requirements: Windows Native Domain functional levels.
  - Map Java JEE role to "Employees".
  - Enable referrals.

    For WebSphere Application Server Version 6.1, use federated repositories, specifically:
    - Use a federated realm.
    - Add the Active Directory top-level domain controller to the repository. Do not add sub-domain controllers, as this will result in multiple matches when searches for userids occur, and logins will fail.
    - Select "Support referrals to other LDAP servers" = "follow".
  - Enable nested groups.

Tip: This option offers the optimal approach when using WebSphere Application Server Versions 6.1 or later. Before WebSphere Application Server version 6.1, referrals were not officially supported.

Summary

The following table summarizes how to find group membership within a Microsoft Active Directory forest.

Table 57. Finding group membership.. The following table identifies group membership levels supported in a Microsoft Active Directory forest.

Group Membership	Map Java EE Roles To	Bind to Which LDAP	Enable	Supported in WebSphere Application Server Version	Comments
Global Groups	Collection of global groups	Top domain controller using port 389/636	Referrals	• Federated repositories in WebSphere Application Server	
Universal groups	Universal groups	Any Global catalog, using port 3268		All	
Global groups in universal groups	Universal groups	Top domain controller using port 389/636	referrals, nesting	• Federated repositories in WebSphere Application Server	Cannot use Windows mixed domain functional level

Configuring to use objectCategory attribute

A federated repository uses the `objectCategory` attribute by default for Active Directory user search filters. You can ensure that the federated repository is configured to use the objectCategory attribute. For example, the federated repositories configuration file, `wimconfig.xml`, should be as shown in the following example:

```
<supportedLDAPEntryType name="user" searchFilter="(objectCategory=user)"...>
<supportedLDAPEntryType name="Group" searchFilter="(objectCategory=Group)"...>
```

Configure the user filter and group filter (advanced properties) like the following example:

```
User Filter: (&(sAMAccountName=%v)(objectCategory=user))
Group Filter: (&cn=%v)(objectCategory=group)
```

Follow the following instructions from the administrative console to complete the search filter with the objectCategory attribute.

1. Click **Security > Global Security**.
2. Under Available realm definitions, select **Federated repositories**, and then **Configure**. In a multiple security domain environment, click **Security domains >** ***domain_name***. Under Security Attributes, expand User Realm, and click **Customize for this domain**. Select the Realm type as **Federated repositories** and then click **Configure**.
3. Under Related items, click **Manage repositories**.
4. Select **Forest > LDAP entity types > PersonAccount**. Under General Properties, find the **Search filter** box.
5. Fill in the search filter.

   ```
   (objectCategory=user)
   ```

gotcha: When you select any of these scenarios to use, consult the appropriate Microsoft Active Directory information to completely understand any implications the scenarios might have on your configuation planning.

Authenticating users with LDAP registries in a Microsoft Active Directory forest

Authenticating a user across multiple repositories, or across a distributed Lightweight Directory Access Protocol (LDAP) repository, such as a Microsoft Active Directory forest can be challenging. In any search of the whole user registry, if there is more than one match at run time, authentication fails because of ambiguous match results.

Before you begin

In any multiple Microsoft Active Directory domain environment, the WebSphere Application Server administrator must consider that the default unique ID in the Microsoft Active Directory is the sAMAccountName attribute of a user.

About this task

User IDs are guaranteed to be unique within a single domain. However they are not guaranteed across a tree or a forest. For example, suppose the user ID, *smith*, is added in the forest and in each subdomain. The search for `sAMAccountName=smith` returns three matches. WebSphere Application Server does not authenticate this user when there is more than one possible match in the registry.

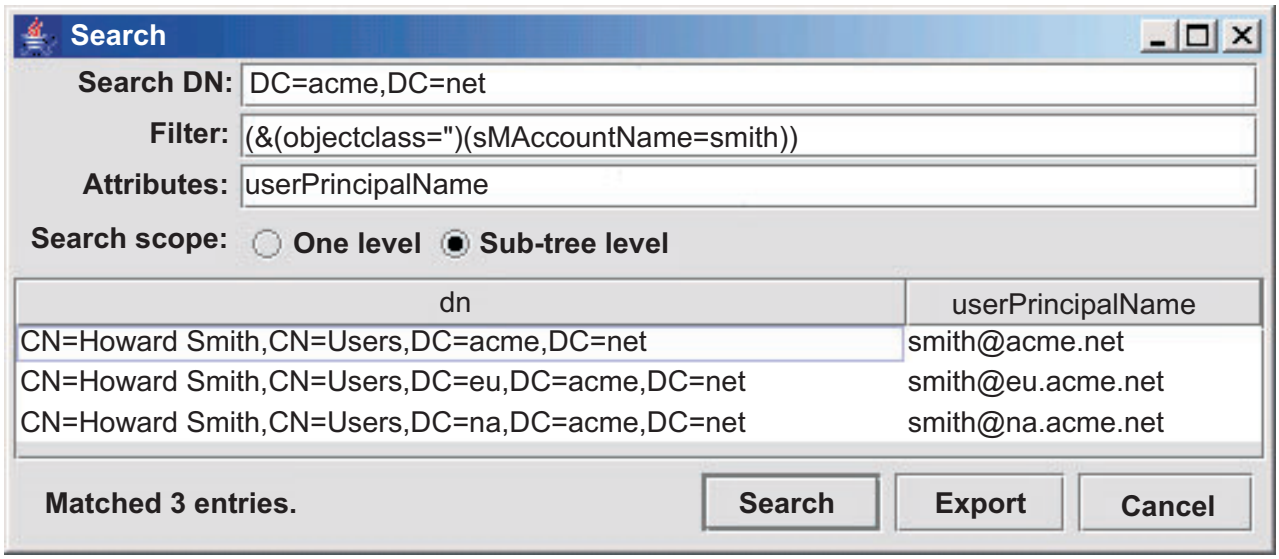


Figure 32. Forest search strategy.. Search illustration of a non-unique sAMAccountName across the entire forest.

You can mitigate this condition by changing the user filter to be based on the userPrincipalName attribute of the user, which is unique across the forest, instead of being based on their sAMAccountName attribute. However, users must then know to log in using their userPrincipalName, which they might not know.

The specific procedure to establish a user filter on a LDAP user registry depends on the type of LDAP registry. The following examples illustrate a procedure for a stand-alone LDAP registry and a procedure for a federated repository registry.

Procedure

1. **Establish a user filter on a stand-alone LDAP registry:** You can set the user filter on the Advance Lightweight Access Protocol (LDAP) user registry settings page to search for userPrincipalName instead of sAMAccountName value.

   For example:

   ```
   (&(objectClass=user)(userPrincipalName=%w))
   ```

2. **Establish a user filter on a federated repositories registry:** You can change the log-in property in the LDAP repository to `uid;cn`, for example, by using the administrative console.
   a. Click **Security > Global security**.
   b. Under Available realm definitions, select **Federated repositories**, and then **Configure**. In a multiple security domain environment, click **Security domains > *domain_name***. Under Security Attributes, expand User Realm, and click **Customize for this domain**. Select the Realm type as **Federated repositories** and then click **Configure**.
   c. Under Related items, click **Manage repositories**.
   d. Click **Add > LDAP repository**.
   e. Under General Properties, add the following information:

      Repository identifier
      forest

      Directory type
      Microsoft WIndows Server 2003 Active Directory

      Primary host name
      forest.acme.net

      Port 389

Failover server used when primary is not available
None

Bind distinguished name
cn=wasbind, CN=Users, DC=ib

Bind password
****

Login properties
uid;cn

3. Click **OK** and **Save** to save the changes to the master configuration.
4. On the LDAP repository configuration page, under Additional properties, click **LDAP attributes**.
5. Click **Add > Supported**.
6. In the **Name** field, enter `userPrincipalName`.
7. In the **Property name** filed, enter `cn`.
8. In the **Entity types** field, enter `PersonAccount`.
9. Click **OK** and **Save** to save the changes to the master configuration.
10. Locate the `{WAS_HOME}\profiles\{profileName}\config\cells\{cellName}\wim\config\wimconfig.xml` or `profile_root/conf/cells/<cell>/wim/config/wimconfig.xml` file in the deployment manager configuration.
11. Edit the `wimconfig.xml` file.
    a. Find the `<config:attributeConfiguration>` attribute in the file.
    b. Add the following lines:

```
<config:attributes name="userPrincipalName" propertyName="cn">
<config:entityTypes>PersonAccount</config:entityTypes>
</config:attributes>
```

12. Save the `wimconfig.xml` file.
13. Run the `profile_root/bin/syncNode.bat` or `profile_root/syncNode.bar/sh` script on all of the nodes in the configuration.

Results

gotcha: When you select any of these scenarios, consult appropriate Microsoft Active Directory information to completely understand any implications the scenarios might have on your configuation planning.

SAML web single sign-on

Security Assertion Markup Language (SAML) is an OASIS open standard for representing and exchanging user identity, authentication, and attribute information. SAML is fast becoming the technology of choice to provide cross-vendor single sign-on (SSO) interoperability.

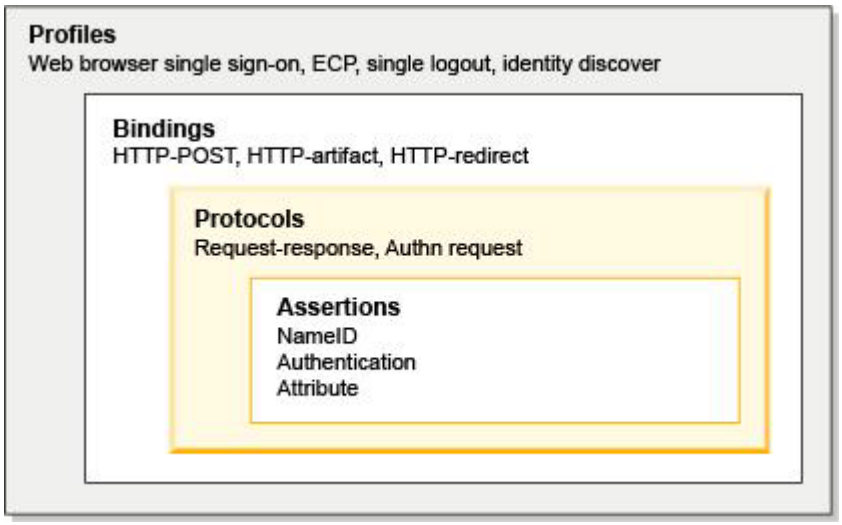
A SAML assertion is an XML-formatted token that is used to transfer user identity and attribute information from the identity provider of a user to a trusted service provider as part of the completion of a single sign-on request. A SAML assertion provides a vendor-neutral means of transferring information between federation business partners.

WebSphere Application Server supports Assertion Markup Language (SAML) web single sign-on, and acts as a SAML service provider. A web user authenticates to a SAML identity provider, which produces an SAML assertion, and WebSphere SAML service provider consumes the SAML assertion to establish a security context for the web user.

As a protocol, SAML has three versions: SAML 1.0, SAML 1.1, and SAML 2.0. SAML 2.0 is an enhancement to the previous SAML 1.x specifications, but is not backwards compatible.

SAML 2.0 defines several request-response protocols, which all correspond to the action being communicated in the message. These protocols are HTTP-redirect based and involve the user's browser. SAML 2.0 has defined several binding options, HTTP redirect, HTTP POST, HTTP artifact, and SOAP. These options specify the way in which messages can be transported. SAML 2.0 HTTP POST enables SAML protocol messages to be transmitted within an HTML form using base64-encoded content. SAML 2.0 HTTP POST enables the SAML provider and consumer to communicate using an HTTP user agent as an intermediary. HTTP POST is sometimes called Browser POST, particularly when used in single sign-on operations. SAML 2.0 Web Browser SSO Profile is defined to support web single sign-on. A web user either accesses a resource at a service provider, or accesses an identity provider such that the service provider and desired resource are understood or implicit. The web user authenticates to the identity provider, which then produces an authentication assertion, and the service provider consumes the assertion to establish a security context for the web user.

The following image shows the SAML SSO overview:



Refer to the specifications and standards for more information.

SAML single sign-on scenarios, features, and limitations

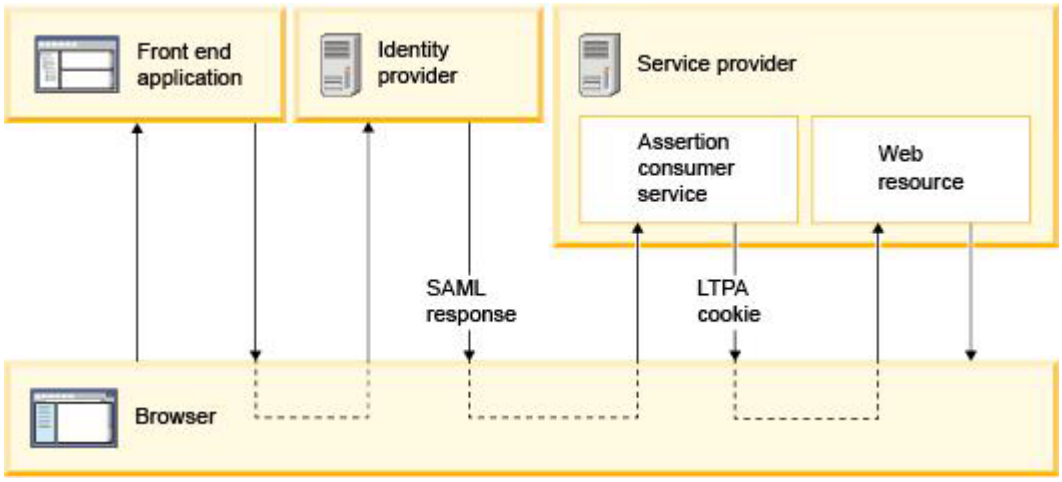
Security Assertion Markup Language (SAML) is an OASIS open standard for representing and exchanging user identity, authentication, and attribute information. SAML is fast becoming the technology of choice to provide cross-vendor single sign-on (SSO) interoperability.

The WebSphere Application Server SAML service provider (SP) supports SAML 2.0 Identity Provider (IdP) initiated single sign-on (SSO). WebSphere IdP initiated SSO service is implemented as a Trust Association Interceptor, and can be described as follows:

1. User accesses a front end web application that can reside on the IdP, SP, or elsewhere.
2. Front end web application redirects user to IdP and user authenticates to IdP.
3. IdP redirects user to Assertion Consumer Service (ACS) in SP by sending SAML response over HTTP POST inside a hidden form.
4. SP processes SAML response and creates WebSphere security context.
5. SP adds LTPA cookie to HTTP response and redirects request to web resource or business application.
6. WebSphere Application Server intercepts request, and maps LTPA cookie to security context and authorizes user access to the requested web resource.

7. WebSphere Application Server sends HTTP response back to user.

The following images shows the SAML SSO flow:



The SAML SSO features include the following:

- The WebSphere SAML service provider supports single sign-on with multiple identity providers.
- The WebSphere SAML service provider supports options for identity assertion and mapping the assertion identity to the user registry of the service provider.
- The WebSphere SAML service provider can map or assert SAML token attributes to the realm, principal, unique Id, and group into the service provider security context.
- The WebSphere SAML service provider provides a plug point to allow for customized identity mapping.
- The WebSphere SAML service provider has an option to retrieve the group membership of the identity from the registry of the service provider and populate the security context.
- The WebSphere SAML service provider provides an IdP selection filter to route the request back to the proper IdP if the request did not come from the IdP.
- The WebSphere SAML service provider supports both RSA-SHA1 and RSA-SHA256 signature algorithms.
- The WebSphere SAML service provider preserves the SAML token in the subject of the service provider for access by the application, and makes it available for a downstream authenticated Enterprise JavaBean (EJB) or Web Service call.
- The WebSphere SAML service provider allows a business application URL to act as an AssertionConsumerService URL, so the IdP can send a SAMLResponse directly to the business application URL.
- The WebSphere SAML trust association interceptor (TAI) allows auditing of key SAML assertions, including Issuer and NameID.

The following feature highlights and best practices apply to the SAML SSO features:

- Assertion consumer service (ACS) in WebSphere SAML service provider:

  ACS is a secured servlet that accepts a SAML protocol message and establishes the security context. An ACS URL has a predefined ContextRoot as `samlsps`, and a URL has the following format:

  ```
  https://<host name>:<port>/samlsps/<any uri pattern>
  ```

  The SAMLResponse received by the ACS will be intercepted by TAI, and upon successful validation, the request is redirected to the target application service.

  Any business service that implements the POST method can act as an ACS. Using a target business servlet as an ACS is preferred, as it reduces one round trip between the browser and the service provider server.

- Multiple security domain support:

  An ACS is deployed in an application security domain, and it is expected that the ACS reside in the same security domain as the business application. If the ACS and target business application (RelayState) are in different security domains, the following are some recommended options:

  – Process the SAMLResponse in the security domain of the ACS.
  – Reconfigure the ACS to have the same domain as the business application.
  – Use the target business service as the ACS.
- Multiple single sign-on partners:

  The WebSphere SAML TAI supports multiple ACS and IdP single sign-on (SingleSignOnService) partners. One SSO partner is defined as one ACS URL, and multiple SingleSignOnServices. With the existence of multiple SSO partners, each SSO partnership is uniquely identified by an ACS URL.

  Each SSO partner can have its own validation rules, mapping rule from assertion to subject, or a rule to start the SSO with its own IdP. For example, one SSO partner can handle ID assertion, which consists of generating a WebSphere platform subject without calling into the user registry. Another SSO partner can perform a local user registry lookup. Another example is that one SSO partner handles SSO with one IdP, and another SSO partner handles SSO with a different IdP.
- Bookmark style SSO and TAI filter:

  Consider a bookmark style SSO which traditionally fits into an SP-initiated SSO. The user accesses the business application without authenticating to the IdP first. The WebSphere SAML TAI can be configured to initiate an SSO. Each SSO partner configuration contains an IdP login application and a routing filter. Each filter defines a list of selection rules that represent conditions that are matched against the HTTP request to determine whether or not the HTTP request is selected for an SSO partner. The filter rule is a combination of HTTP request header, referrer data, and target application name. The WebSphere SAML TAI runtime environment checks the user request against all filter rules to uniquely identify the SSO partner, and redirects the request to the selected IdP login application. The TAI filter allows an IdP-initiated SSO to provide similar functionality as the combination of an SP-initiated SSO and an IdP discovery service.
- Identity mapping and security context management:

  The WebSphere SAML TAI provides a rich and flexible identity mapping, and can be classified as follows:

  – Identity assertion: Map the SAML assertion to the WebSphere platform subject without a local registry. Typical ID assertion scenarios include:
    - Default: use NameID as principal, issuer as realm, selected attribute as group members.
    - Customized: configure SAML attribute as principal, realm, accessID, and group members.
  – Map NameID from the IdP against the user registry of the service provider, and build the subject from the registry. The following scenarios are supported:
    - Directly map the SAML NameId to the local registry.
    - Plugin point for custom mapping, followed by using a new user to build the subject.
    - Map NameID to the user registry, and fall back to ID assertion.
  – Combination of ID assertion and local registry:

    In addition to ID assertion, TAI searches parent groups of the asserted groups in the user registry of the service provider, and includes the parent groups into the subject. For example, authorization is granted to parent groups, but the identity provider does not know the parent group names.

WebSphere Application Server supports IdP initiated SAML web SSO only.

The following specifications or scenarios are out of scope:

- Enhanced Client or Proxy (ECP) Profile
- Identity Provider Discovery Profile
- Single Logout Profile

- Name Identifier Management Profile
- Artifact Resolution Profile
- Assertion Query/Request Profile
- Name Identifier Mapping Profile
- SAML Attribute Profiles

Enabling your system to use the SAML web single sign-on (SSO) feature

Before you begin

This task assumes that you are familiar with the SAML SSO feature.

About this task

Before you can use the SAML Web SSO feature, you must install the SAML Assertion Consumer Service (ACS) and enable SAML TAI. If you are planning to use your business application as the SAML ACS application, you do not need to install the SAML ACS application in the first step. You should instead specify the URL of the business application for the `acsUrl` value.

Procedure

1. Install the SAML ACS application.
   a. Navigate to the `app_server_root/bin` directory.
   b. Run the `installSamlACS.py` script. For example:

      ```
      wsadmin -f installSamlACS.py install <nodeName> <serverName>
      ```

      or

      ```
      wsadmin -f installSamlACS.py install <clusterName>
      ```

      where `nodeName` is the node name of the target application server, `serverName` is the server name of the target application server, and `clusterName` is the name of the application server cluster.
2. Enable SAML TAI. You can enable SAML TAI by using either the `wsadmin` command utility or the administrative console.
   - Enable SAML TAI using the `wsadmin` command utility.

   a. Start the WebSphere Application Server.
   b. Start the `wsadmin` command utility from the `app_server_root/bin` directory by entering the command: `wsadmin -lang jython`.
   c. At the wsadmin prompt, enter the following command: `AdminTask.addSAMLTAISSO('-enable true -acsUrl https://<hostname>:<sslport>/samlsps/<any URI pattern string>')` where `hostname` is the host name of the system where WebSphere Application is installed and `sslport` is the Web server SSL port number (WC_defaulthost_secure).
   d. Save the configuration by entering the following command: `AdminConfig.save()`.
   e. Exit the `wsadmin` command utility by entering the following command: `quit`.
   f. Restart the WebSphere Application Server.

   - Enable SAML TAI using the administrative console.

   a. Log on to the WebSphere Application Server administrative console.
   b. Click **SecurityGlobal security**.
   c. Expand **Web and SIP security** and click **Trust association**.
   d. Under the **General Properties** heading, select the **Enable trust association** check box and click **Interceptors**.

e. Click **New** and enter `com.ibm.ws.security.web.saml.ACSTrustAssociationInterceptor` in the **Interceptor class name** field.

f. Under **Custom properties**, fill in the following custom property information: Name: `sso_1.sp.acsUrl` and Value: `https://<hostname>:<sslport>/samlsps/<any URI pattern string>` where `hostname` is the host name of the system where WebSphere Application is installed and `sslport` is the Web server SSL port number (WC_defaulthost_secure).

g. Click **New** and enter the following custom property information: Name: `sso_1.sp.idMap` and Value: `idAssertion`.

h. Click **OK**.

i. Go back to **SecurityGlobal security** and click **Custom properties.**

j. Click **New** and define the following custom property information under **General properties**: Name: `com.ibm.websphere.security.DeferTAItoSSO` and Value: `com.ibm.ws.security.web.saml.ACSTrustAssociationInterceptor`.

Note: If this custom property already exists, edit its value to add `com.ibm.ws.security.web.saml.ACSTrustAssociationInterceptor`.

k. Click **New** and define the following custom property information under **General properties**: Name: `com.ibm.websphere.security.InvokeTAIbeforeSSO` and Value: `com.ibm.ws.security.web.saml.ACSTrustAssociationInterceptor`.

l. Click **OK**.

m. Restart WebSphere Application Server.

Results

The SAML TAI is now enabled for WebSphere Application Server.

What to do next

After enabling the SAML Web SSO feature, you must configure WebSphere Application Server as a service provider (SP) partner to participate in the IdP-initiated single sign-on scenarios with other identity providers.

Configuring single sign-on (SSO) partners

Before you begin

This task assumes that you have enabled the SAML Web SSO feature.

About this task

Before you can use the WebSphere Application Server as a service provider partner to identity providers for IdP-initiated single sign-on, you need to establish partnerships between the WebSphere Application Server SAML service provider and external SAML identity providers.

Procedure

1. Add an identity provider to the WebSphere Application Server SAML service provider for single sign-on. To use the WebSphere Application Server SAML service provider for single sign-on with an identity provider, you need to add the identity provider as a partner. You can add an identity provider as a partner either by importing the metadata of the identity provider, or by using manual steps.
   - Add an identity provider using metadata of the identity provider.

   a. Start the WebSphere Application Server.

   b. Start the `wsadmin` command-line utility from the `app_server_root/bin` directory by entering the command: `wsadmin -lang jython`.

c. At the wsadmin prompt, enter the following command: `AdminTask.importSAMLIdpMetadata('-idpMetadataFileName <IdPMetaDataFile> -ssold 1 -ipdld 1 -signingCertAlias <idpAlias>')` where `IdpMetaDataFile` is the full path name of the IdP metadata file, and `IdpAlias` is any alias name that you specify for the imported certificate.

d. Save the configuration by entering the following command: `AdminConfig.save()`.

e. Exit the `wsadmin` command utility by entering the following command: `quit`.

f. Restart the WebSphere Application Server.

- Manually add an identity provider to the WebSphere Application Server SAML service provider.

  The minimum requirement to configure the WebSphere Application Server SAML service provider as an SSO partner to an identity provider is to import the SAML token signer certificate from the identity provider to the trust store of the service provider. The service provider can be configured to work with multiple identity providers. For each identity provider, you must import the SAML token signer certificate.

  You can import the certificate used by an IdP to sign the SAML token by using either the administrative console or the `wsadmin` command-line utility.

- Import the SAML token signer certificate using the administrative console.

a. Log on to the WebSphere Application Server administrative console.

b. Click **Security** > **SSL certificate and key management** > **Key stores and certificates** > **NodeDefaultTrustStore** > **Signer certificates**. Use **CellDefaultTrustStore** instead of **NodeDefaultTrustStore** for a deployment manager.

c. Click **Add**.

d. Fill in the certificate information.

e. Click **Apply**.

- Import the SAML token signer certificate using the `wsadmin` command-line utility.

a. Start the WebSphere Application Server.

b. Start the `wsadmin` command-line utility from the `app_server_root/bin` directory by entering the command: `wsadmin -lang jython`.

c. At the wsadmin prompt, enter the following command: `AdminTask.addSignerCertificate('[-keyStoreName NodeDefaultTrustStore -certificateFilePath <certFile> -base64Encoded true -certificateAlias <certAlias>]')` where `certFile` is the full path name of the certificate file and `certAlias` is the alias of the certificate. Use `CellDefaultTrustStore` instead of `NodeDefaultTrustStore` for a deployment manager.

d. Save the configuration by entering the following command: `AdminConfig.save()`.

e. Exit the `wsadmin` command utility by entering the following command: `quit`.

2. Add IdP realms to the list of inbound trusted realms. For each Identity provider that is used with your WebSphere Application Server service provider, you must grant inbound trust to all the realms that are used by the identity provider.

   You can grant inbound trust to the identity providers using either the administrative console or the `wsadmin` command utility.

   - Add inbound trust using the administrative console.

   a. Click **Global security**.

   b. Under user account repository, click **Configure**.

   c. Click **Trusted authentication realms - inbound**.

   d. Click **Add External Realm**.

   e. Fill in the external realm name.

   f. Click **OK** and **Save changes to the master configuration**.

   - Add inbound trust using the `wsadmin` command-line utility.

a. To add a single identity provider to the inbound trust, use the following command: `AdminTask.addTrustedRealms('[-communicationType inbound -realmList <realmName>]')` where `realmName` is the name of the realm that needs to be granted inbound trust.

b. To add a list of realms to the inbound trust, use the following command: `AdminTask.addTrustedRealms('[-communicationType inbound -realmList <realm1|realm2|realm3>]')` where `realm1`, `realm2`, and `realm3` are the realms that need to be added as trusted realms.

3. Add the WebSphere Application Server SAML service provider to the identity providers for SSO.

   Each identity provider that is used with your WebSphere Application Server service provider needs to be configured to add the service provider as an SSO partner. The procedure for adding the service provider partner to an identity provider depends on the specific identity provider. Refer to the documentation of the identity provider for instructions on how to add a service provider partner for SSO.

   You can either export the WebSphere Application Server service provider metadata, and import it to the identity provider, or manually configure the identity provider to add the service provider.

   To add the service provider as a federation partner to an identity provider, you must provide the URL of the Assertion Consumer Service (ACS) of the service provider, which is the `-acsUrl` parameter used when enabling the SAML trust association interceptor (TAI).

   If an identity provider can use a metadata file to add the service provider as a federation partner, you can use the following `wsadmin` command-line utility command to export the service provider metadata:

   ```
   wsadmin -lang jython
   AdminTask.exportSAMLSpMetadata(-spMetadataFileName /tmp/spdata.xml -ssoId 1')
   ```

   This command creates the `/tmp/spdata.xml` metadata file.

   If the SAML token is encrypted, you must provide the public key certificate that you want the identity provider to use for encrypting the SAML token, and the certificate must exist in the WebSphere Application Server default KeyStore before performing an export.

4. Configure the WebSphere Application Server security context. The WebSphere Application Server service provider intercepts a SAML protocol message from the identity provider and establishes the security context. The security context is created by mapping the SAML assertion. The security context mapping in the service provider is very flexible and configurable. The following is a list of available mapping options:

   - ID assertion

     You can map the SAML assertion to the WebSphere Application Server platform Subject without using a local registry and this is the default behavior. In this default implementation, the SAML `NameID` is mapped to the principal, the issuer is mapped to the realm, and selected attributes can be mapped to group members. ID assertion can be further customized. For example, you can configure a SAML attribute as a principal, realm, accessId, or a list of group members. You can also configure `NameQualifier` from `NameID` as a realm, or use a predefined realm name.

   - localRealm

     You can configure the WebSphere Application Server service provider to map the `NameID` from a SAML assertion to the local registry of the service provider, and build the subject from the registry. With this option, you can directly search the SAML `NameID` against the registry, or use a plugin point for custom mapping of the assertion, and then use the new mapped ID to build the subject from the registry.

   - localRealmThenAssertion

     This option allows you to map the `NameID` to the registry, and fall back to ID assertion if the `NameID` cannot be mapped to the registry.

   - ID assertion using groups

     This option combines ID assertion and local registry and allows you to reevaluate group membership while doing ID assertion. Consider a SAML assertion from a partner lab, containing user `Joe` with a group attribute of `X-ray Techs`. At the service provider, the group `X-ray Techs` is a

subgroup of group `Technicians`, but `Joe` is not necessarily in the user registry of the service provider. The authorization policy of the service provider application allows access to the `Technicians` group. To achieve this, the SAML TAI needs to look up the asserted groups `X-ray Techs` in the registry and then include the parent groups `Technicians` in the Subject.

When doing ID assertion to create a security context, a custom security realm is chosen. You must explicitly add the custom realm as a trusted realm. In a default ID assertion implementation, the SAML issuer name is used as the security realm. You must explicitly add the issuer name to the list of inbound trusted authentication realms in current user registry. After adding the custom realm to the inbound trusted realms, you are ready to do role mapping with this custom realm.

To add a custom realm as a trusted realm, see the *Add IdP realms to the list of inbound trusted realms* step.

Results

Your WebSphere Application Server is now configured as a service provider partner for IdP-initiated SSO.

What to do next

For additional configuration options for your service provider, see the *SAML web SSO TAI custom properties* topics for a complete list of SAML TAI custom properties.

SAML web single sign-on (SSO) trust association interceptor (TAI) custom properties

The following tables list the custom properties for the Security Assertion Markup Language (SAML) trust association interceptor (TAI). You can define these properties in the custom properties panel for the SAML TAI using the administrative console.

To assign unique property names that identify each possible single sign-on (SSO) service provider (SP) partner, an `sso_<id>` is embedded in the property name and used to group the properties that are associated with each SSO partner. The `sso_<id>`s are numbered sequentially for each SSO service provider partner.

The SAML TAI custom properties can be grouped into three categories:

1. Global properties - these properties are applicable to all SSO partners that are configured for the SAML TAI.
2. IdP properties - these properties are applicable to identity providers that are configured for the SAML TAI. To assign unique property names that identify each identity provider partner, an `idp_<id>` is embedded in the property name and used to group the properties that are associated with each SSO IdP partner.
3. Service provider properties - these properties are applicable to a service provider and are grouped together for each SSO service provider partner under a unique `sso_<id>`.

The following table describes the global SAML TAI custom properties:

Table 58. Global SAML TAI custom properties

Property name	Values	Description
`targetUrl`	You can specify any URL value.	This property is overridden by `sso_<id>.sp.targetUrl`. This is the default target URL after successful validation of the `SAMLResponse` when there is no `RelayState` received from the IdP.

Table 58. Global SAML TAI custom properties (continued)

Property name	Values	Description
useRelayStateForTarget	You can specify one of the following values: • true (Default) • false	This property is overridden by sso_<id>.sp. useRelayStateForTarget. This is used to indicate if the RelayState should be used as the target URL.
allowedClockSkew	You can specify any positive number. The default is five minutes.	This property is overridden by sso_<id>.sp. allowedClockSkew. This is used to specify the allowed clock skew in minutes when validating the SAML token.
enforceTaiCookie	You can specify one of the following values: • true (Default) • false	This property is overridden by sso_<id>.sp. enforceTaiCookie. This is used to indicate if the SAML TAI should check if an LTPA cookie is mapped to a subject created for the SSO partner.
replayAttackTimeWindow	You can specify any integer value. The default value is 30.	This property specifies the time, in minutes, within which the second request is rejected if two identical SAML tokens are received by the TAI. See also sso_<id>.sp. preventReplayAttack.

The following table describes the IdP SAML TAI custom properties:

Table 59. IdP SAML TAI custom properties

Property Name	Values	Description
sso_<id>.idp_<id>.SingleSignOnUrl	You can specify any URL value.	This custom property specifies the URL of the SSO service of the IdP.
sso_<id>.idp_<id>.allowedIssuerDN	This custom property does not have a default value.	This custom property specifies the name of the Issuer who is allowed to sign the SAML token sent by the IdP. If the SAML token is not signed by this issuer, the token is rejected.
sso_<id>.idp_<id>.allowedIssuerName	This custom property does not have a default value.	This custom property specifies the value of the <saml:Issuer> Issuer element in the SAML token. The SAML token received from the IdP is rejected if the Issuer in the token does not match this value.

The following table describes the service provider SAML TAI custom properties:

Table 60. Service provider SAML TAI custom properties

Property Name	Values	Description
sso_<id>.sp.acsUrl	This property does not have a default value. You can specify one of the following values: • URL of the assertion consumer service (ACS): https://<hostname>:<sslport>/samlsps/<any URI pattern String> • URL of the business application	This is the only required property for each sso_<id>. It specifies the URL of the ACS or business application.
sso_<id>.sp.EntityID	By default, this property is set to the value of sso_<id>.sp.aclUrl.	This property is used to verify AudienceRestriction in the SAML assertion.
sso_<id>.sp.targetUrl	This property does not have a default value.	This property specifies the URL of the target application. It is used when RelayState is not present in the client request.
sso_<id>.sp.useRelayStateForTarget	You can specify one of the following values: • true (Default) - specify this value is you want to use the value of RelayState in the client request as the URL of the target application. • false - specify this value if you want to use the value of sso_<id>.sp.targetUrl as the URL of the target application.	This property specifies whether the RelayState value received in the client request should be used as the URL of the target application or not. If this property is set to false, the sso_<id>.sp.targetUrl property is used as the URL of the target application.
sso_<id>.sp.login.error.page	This property does not have a default value.	This property specifies the error page, IdP login page, or custom mapping class to which an unauthenticated client request is redirected to.
sso_<id>.sp.acsErrorPage	This property does not have a default value.	This property is used to override sso_<id>.sp.login.error. page.
sso_<id>.sp.allowedClockSkew	This property does not have a default value.	This property specifies, in minutes, the time that is added to the token expiration time of the SAML token sent by the IdP.
sso_<id>.sp.trustStore	This property does not have a default value.	This property specifies the truststore for validating the SAML signature. It specifies the name of a managed keystore.
sso_<id>.sp.trustAnySigner	You can specify one of the following values: • false (Default) - the signer certificate is verified for trust validation • true - any signer certificate is trusted without trust validation	This property specifies how the signer certificate of the SAML token is verified for trust validation. If this property is set to true, any signer certificate is trusted.
sso_<id>.sp.keyStore	This property does not have a default value.	This property specifies the keystore that contains the private key for decrypting the encrypted SAML assertion.

Table 60. Service provider SAML TAI custom properties (continued)

Property Name	Values	Description
sso_<id>.sp.keyName	This property does not have a default value.	This property specifies the key name for decrypting the SAML assertion.
sso_<id>.sp.keyPassword	This property does not have a default value.	This property specifies the key password for decrypting the SAML assertion.
sso_<id>.sp.keyAlias	This property does not have a default value.	This property specifies the key alias for decrypting the SAML assertion.
sso_<id>.sp.wantAssertionsSigned	You can specify one of the following values: • true (Default) - the service provider requires the IdP to sign the SAML assertion • false - the SAML assertion is not required to be signed by the IdP	If this property is set to false, the SAML assertion is not required to be signed and the signature is not validated.
sso_<id>.sp.preserveRequestState	You can specify one of the following values: • true (Default) - the client state is saved and restored when it is redirected to the IdP login • false - the client state is not saved	When the service provider redirects the client request to the IdP login, this property specifies whether the client state needs to be saved and restored after the client request is completed.
sso_<id>.sp.enforceTaiCookie	You can specify one of the following values: • true (Default) • false	This property is used to indicate if the SAML TAI should check if an LTPA cookie is mapped to a subject created for the SSO partner.
sso_<id>.sp.realmName	This can be any string value. By default, this property is set to the SAML Issuer name.	This property specifies any SAML attribute and is used in conjunction with realmNameRange. The value of this attribute is used as the subject realm. If this realm does not exist in the list of realms specified by realmNameRange, the realm is rejected.
sso_<id>.sp.realmNameRange	This property has no default value.	This property specifies a list of allowed realm names and is used in conjunction with realmName. See the description of sso_<id>.sp.realmName.
sso_<id>.sp.principalName	This can be any string value. By default, this property is set to the Subject NameID.	This property specifies any SAML attribute. The value of this attribute is used as the subject principal.
sso_<id>.sp.uniqueId	This can be any string value. By default, this property is set to the Subject NameID.	This property specifies any SAML attribute. The value of this attribute is used as the subject uniqueId.
sso_<id>.sp.groupName	This property does not have a default value.	This property specifies any SAML attribute. The value of this attribute is used as groups in the subject.

Table 60. Service provider SAML TAI custom properties (continued)

Property Name	Values	Description
sso_<id>.sp.defaultRealm	You can specify one of the following values: • IssuerName (Default) - use the SAML token Issuer as the default realm • NameQualifier - use the SAML token NameQualifier as the default realm	This custom property specifies whether the Issuer or the NameQualifier from the SAML assertion is used as the default realm.
sso_<id>.sp.useRealm	This property does not have a default value.	This property specifies a realm name and is used to override the default realm. This property also overrides the realmName property.
sso_<id>.sp.idMap	You can specify one of the following values: • idAssertion (Default) - the user specified in the SAML assertion is not checked in the local registry • LocalRealm - the SAML token user is verified in the local user registry • localRealm - is the user is found in the local registry, IDAssertion is used	This property specifies how the SAML token is mapped to the subject.
sso_<id>.sp.groupMap	You can specify one of the following values: • localRealm - specify this value to map the SAML token groups to groups and parent groups found in the local user registry • AddGroupsFromLocalRealm - specify this value to map the SAML token groups to groups, even if the groups do not exist in the local user registry	This property is used with IDAssertion and specifies how the SAML token is mapped to the groups.
sso_<id>.sp.userMapImpl	This property does not have a default value.	This property specifies the name of a custom user mapping module class. It is used to map a user ID in the SAML token to another user ID that exists in the local user registry.
sso_<id>.sp.X509PATH	This property does not have a default value.	This property specifies the certificate store that is used for the intermediary certificates used in validating the SAML signature.
sso_<id>.sp.CRLPATH	This property does not have a default value.	This property specifies the certificate store that is used for certificate revocation lists (CRLs) used in validating the SAML signature.

Table 60. Service provider SAML TAI custom properties (continued)

Property Name	Values	Description
sso_<id>.sp.filter	This property does not have a default value.	This property is used to specify a condition that is checked against the HTTP request, to determine whether or not the HTTP request is selected for a SAML web SSO partner. See the *SAML TAI filter property* section for more information on this property.
sso_<id>.sp.preventReplayAttack	You can specify one of the following values: • true (Default) • false	This property is used to specify whether the SAML TAI should prevent two identical SAML tokens sent in client requests. This property is used in conjunction with the global property replayAttackTimeWindow.
sso_<id>.sp.trustedAlias	This property does not have a default value.	If this property is specified, only the key specified by this alias is used to validate the signature in the SAML assertion. If the signature in the incoming SAML assertion of the SAMLResponse does not include the KeyInfo element, specify this property to resolve the KeyInfo element.

SAML TAI filter property

The sp.filter SAML TAI filter property is used when a client invokes a protected service provider application directly, without authenticating to the IdP. The filter property is usually used in conjunction with the sp.login.error.page property to redirect an unauthenticated client request to the URL address specified by the sp.login.error.page property.

The filter property specifies a set of conditions that are compared against the HTTP request of the client to select a SAML web SSO service provider partner for processing the HTTP request. Each condition is specified by three elements:

- input required - the input element usually specifies an HTTP header name, but request-url and remote-address can also be used as special elements
- operator - the operator element specifies one of the following values: ==, !=, %=, ^=, <, and >
- comparison value - this element usually specifies a string, but IP address ranges are also allowed

The conditions are evaluated from left to right, as specified by the comparison value. If all the filter conditions specified by an SSO service provider partner are met in an HTTP request, the SSO service provider partner is selected for the HTTP request.

The input element identifies an HTTP request header field to extract from the request and its value is compared with the value that is specified in the filter property according to the operator specification. If the header field that is identified by the input element is not present in the HTTP request, the condition is treated as not being met. Any of the standard HTTP request header fields can be used as the input element in the filter condition. Refer to the HTTP specification for the list of valid headers.

In addition to the standard HTTP header fields, the following two special input elements can be used in the filter property:

- request-url - the comparison value of this input is compared against the URL address that is used by the client application to make the request

- remote-address - the comparison value of this input is compared against the TCP/IP address of the client application that sent the HTTP request

Examples

In the following example, the filter property specifies an HTTP header field `From` as the input with `samluser@xyz.com` as the comparison value and `==` as the operator:

```
sso_1.sp.filter=From==samluser@xyz.com
```

In this case, if a client request contains an HTTP header field `From` with a value of `samluser@xyz.com`, the SAML TAI selects the SSO service provider partner of this `sso_1` filter for processing the client request.

In the following example, the filter property specifies a URL with `ivtlanding.jsp` as the comparison value and `%=` as the operator:

```
sso_2.sp.filter=request-url%=ivtlanding.jsp
```

In this case, if the URL of the target application invoked by the client contains the string `ivtlanding.jsp`, the SAML TAI selects the SSO partner of this `sso_2` filter for processing the client request.

In the following example, the filter property specifies an application name with `DefaultApplication` as the comparison value and `==` as the operator:

```
sso_3.sp.filter=applicationNames==DefaultApplication
```

In this case, if the name of the target application invoked by the client application is `DefaultApplication`, the SAML TAI selects the SSO partner of this `sso_3` filter for processing the client request.

The following table lists the different operators used in the filter property:

Table 61. Filter property operators

Operator	Condition	Example
`==`	This operator specifies an exact match. The input element must be equal to the comparison value.	`From==jones@my.company.com`
`%=`	This operator specifies a partial match. The input element contains the comparison value.	`user-agent%=IE 6`
`^=`	The input element contains one of the comparison values.	`request-url^=urlApp1|urlApp2|urlApp3`
`!=`	The input element does not contain the comparison value.	`request-url!=SPNEGO`
`>`	The input element is greater than the comparison value.	`remote-address>192.168.255.130`
`<`	The input element is less than the comparison value.	`remote-address<192.168.255.135`

Adding SAML web single sign-on (SSO) trust association interceptor (TAI) using the wsadmin command-line utility

About this task

The `addSAMLTAISSO` command adds the Security Assertion Markup Language (SAML) trust association interceptor (TAI) in the security configuration of the WebSphere Application Server.

Procedure

1. Start the WebSphere Application Server.
2. Start the `wsadmin` command utility from the `app_server_root/bin` directory by entering the command: `wsadmin -lang jython`.
3. At the `wsadmin` prompt, enter the following command:

   ```
   AdminTask.addSAMLTAISSO('-enable true -acsUrl https://<hostname>:<sslport>/samlsps/<any URI pattern String>')
   ```

   where `hostname` is the host name of the system on which WebSphere Application Server is installed, and `sslport` is the Web server SSL port number (`WC_defaulthost_secure`).

   You can use the following parameters with this command:

Table 62. addSAMLTAISSO parameters

Parameter	Description
`-acsUrl`	This parameter is required. It specifies the assertion consumer service (ACS) URL.
`-enable`	This parameter specifies whether to enable or disable trust association. You can specify either `true` or `false`.
`-ssoId`	This parameter is optional and is specified as an integer. It is the identifier for the group of custom properties that are defined for the SSO service provider partner. If this parameter is not specified, the next available identifier is used.
`-securityDomainName`	This parameter specifies the name of the security domain of interest and is specified as a String. If a value for this parameter is not specified, the command uses the global security configuration.
`-trustStoreName`	This parameter specifies the truststore name if not using the system default truststore.
`-keyStoreName`	This parameter specifies the keystore name if not using the system default keystore.
`-keyName`	This parameter specifies the key name used to decrypt the encrypted SAML assertion.
`-keyAlias`	This parameter specifies the key alias used to decrypt the encrypted SAML assertion.
`-keyPassword`	This parameter specifies the key password used to decrypt the encrypted SAML assertion.
`-idMap`	This parameter specifies how the SAML token is mapped to the subject. You can specify one of the following values: • idAssertion - the user specifies in the SAML assertion is not checked in the local registry • localRealm - the SAML token user is verified in the local user registry • localRealmThenAssertion - if the user is not found in the local registry, IDAssertion is used

There are additional SAML web SSO TAI custom properties that are not supported by the `addSAMLTAISSO` command, but you can add these custom properties using the `wsadmin` command `configureInterceptor`. For a complete list of the supported SAML TAI properties, see the *SAML web SSO TAI custom properties* topic.

Results

The SAML web SSO TAI is now added for this WebSphere Application Server.

Example

The following example adds the SAML TAI to the global security configuration:

```
AdminTask.addSAMLTAISSO('-enable true -acsUrl https://test1.abc.com:9443/samlsps/acs')
```

The following example adds the SAML TAI SSO service provider partner to the security domain `myDomain1`:

```
AdminTask.addSAMLTAISSO('-securityDomainName myDomain1 -enable true -acsUrl https://test2.xyz.com:9444/samlsps/ac
```

Deleting SAML web single sign-on (SSO) identity provider (IdP) partner using the wsadmin command-line utility

About this task

You can use the `wsadmin` command-line utility to delete an identity provider (IdP) partner in the Security Assertion Markup Language (SAML) web single sign-on (SSO) trust association interceptor (TAI) configuration for WebSphere Application Server.

Procedure

1. Start the WebSphere Application Server.
2. Start the `wsadmin` command utility from the `app_server_root/bin` directory by entering the command: `wsadmin -lang jython`.
3. At the `wsadmin` prompt, enter the following command:

   ```
   AdminTask.deleteSAMLIdpPartner('-ssoID 1 -idpId 1')
   ```

   You can use the following parameters with this command:

Table 63. deleteSAMLIdpPartner parameters

Parameter	Description
`-ssoId`	This parameter is optional if you have only one SSO service provider partner. If you have more than one SSO service provider partner, this parameter is required. It is the identifier for the group of custom properties that are associated with the SSO service provider partner. This parameter is specified as an integer.
`-idpId`	This parameter is required. It specifies the identifier of the IdP that needs to be deleted from the specified SSO service provider partner. This parameter is specified as an integer.
`-securityDomainName`	This parameter specifies the name of the security domain of interest. If a value for this parameter is not specified, the command uses the global security configuration. This parameter is specified as a String.
`-deleteSigningCert`	This parameter is optional. Specify `true` if you want to delete the signing certificate from the trust store. If this alias is referenced by another IdP or service provider, it is not deleted from the trust store. This parameter is specified as a Boolean.

Results

The SAML TAI IdP partner properties have been deleted for this WebSphere Application Server.

Example

The following example deletes the SAML IdP partner 1 of SSO service provider partner 1 from the global security SAML TAI configuration:

```
AdminTask.deleteSAMLIdpPartner('-ssoId 1 -idpId 1')
```

The following example deletes the SAML IdP partner 1 of SSO service provider partner 1 from the security domain `myDomain1`:

```
AdminTask.deleteSAMLIdpPartner('-ssoId 1 -idpId 1 -securityDomainName myDomain1')
```

Deleting SAML web single sign-on (SSO) trust association interceptor (TAI) using the wsadmin command-line utility

About this task

You can use the `wsadmin` command-line utility to delete the Security Assertion Markup Language (SAML) trust association interceptor (TAI) in the security configuration of the WebSphere Application Server.

Procedure

1. Start the WebSphere Application Server.
2. Start the `wsadmin` command utility from the `app_server_root/bin` directory by entering the command: `wsadmin -lang jython`.
3. At the `wsadmin` prompt, enter the following command:

   ```
   AdminTask.deleteSAMLTAISSO()
   ```

   You can use the following parameters with this command:

Table 64. deleteSAMLTAISSO parameters

Parameter	Description
`-ssoId`	This parameter is optional if you have only one SSO service provider partner. If you have more than one SSO service provider partner, this parameter is required. It is the identifier for the group of custom properties that are associated with the SSO service provider partner. This parameter is specified as an integer.
`-securityDomainName`	This parameter specifies the name of the security domain of interest. If a value for this parameter is not specified, the command uses the global security configuration. This parameter is specified as a String.
`-deleteSigningCert`	This parameter is optional. Specify `true` if you want to delete the signing certificate from the trust store. If this alias is referenced by another IdP or service provider, it is not deleted from the trust store. This parameter is specified as a Boolean.

Results

The SAML TAI SSO service provider partner properties have been deleted for this WebSphere Application Server.

Example

The following example deletes the SAML TAI SSO service provider partner 1 from the global security SAML TAI configuration:

```
AdminTask.deleteSAMLTAISSO('-ssoId 1')
```

The following example deletes the SAML TAI SSO service provider partner 1 from the security domain myDomain1:

```
AdminTask.deleteSAMLTAISSO('-ssoId 1 -securityDomainName myDomain1')
```

Exporting SAML web service provider metadata using the wsadmin command-line utility

About this task

You can use the wsadmin command-line utility to export the Security Assertion Markup Language (SAML) trust association interceptor (TAI) service provider metadata to a file.

Procedure

1. Start the WebSphere Application Server.
2. Start the wsadmin command utility from the app_server_root/bin directory by entering the command: wsadmin -lang jython.
3. At the wsadmin prompt, enter the following command:

   ```
   AdminTask.exportSAMLSpMetadata('-spMetadataFileName /tmp/spdata.xml -ssoId 1')
   ```

   You can use the following parameters with this command:

Table 65. exportSAMLSpMetaData parameters

Parameter	Description
-ssoId	This parameter is optional if you have only one SSO service provider partner. If you have more than one SSO service provider partner, this parameter is required. It is the identifier for the group of custom properties that are associated with the SSO service provider partner. This parameter is specified as an integer.
-securityDomainName	This parameter specifies the name of the security domain of interest. If a value for this parameter is not specified, the command uses the global security configuration. This parameter is specified as a String.
-spMetadataFileName	This parameter is required. Specify the fully-qualified file name for the SAML service provider metadata. This parameter is specified as a String.
-wantAssertionsSigned	This parameter is optional. Specify true if you want SAML assertions to be signed. This parameter is specified as a Boolean.
-encryptionMethod	This parameter is optional. It specifies the encryption method. The default value is http://www.w3.org/2001/04/xmlenc#rsa-1_5. This parameter is specified as a String.

Results

The SAML TAI service provider metadata is now exported to the specified file.

Example

The following example exports the SAML service provider metadata of SSO partner 1 from the global security SAML TAI configuration:

```
AdminTask.exportSAMLSpMetadata('-spMetadataFileName /tmp/mySPmetadata.xml -ssoId 1')
```

The following example exports the SAML service provider metadata of SSO service provider partner 1 from the security domain myDomain1:

```
AdminTask.exportSAMLSpMetadata('-spMetadataFileName /tmp/mySPmetadata.xml -ssoId 1 -securityDomainName myDomain1')
```

Importing SAML identity provider (IdP) partner metadata using the wsadmin command-line utility

Before you begin

Before you can use this command, you must configure the Security Assertion Markup Language (SAML) trust association interceptor (TAI) with at least one single sign-on (SSO) partner using the addSAMLTAISSO command. If you create your own trust store, then it must be specified in the sso_<ID>.sp.trustStore entry. If you do not specify the sp.trustStore property, the default truststore is used. All the certificates of the identity provider (IdP) and service provider are saved in the same truststore.

About this task

You can use the wsadmin command-line utility to import the SAML IdP partner to the SAML TAI in the security configuration for WebSphere Application Server. This command will import the following IdP partner data:

- Entity ID
- Signing Certificate
- SingleSignOnService HTTP-POST binding

Note: If any of the above properties are missing, the command logs a warning message.

Procedure

1. Start the WebSphere Application Server.
2. Start the wsadmin command utility from the app_server_root/bin directory by entering the command: wsadmin -lang jython.
3. At the wsadmin prompt, enter the following command:

   ```
   AdminTask.importSAMLIdpMetadata('-idpMetadataFileName /tmp/idpdata.xml
                                    -idpId 1 -ssoId 1 -signingCertAlias idpcert')
   ```

   You can use the following parameters with this command:

Table 66. importSAMLIdpMetaData parameters

Parameter	Description
-ssoId	This parameter is optional if you have only one SSO service provider partner. If you have more than one SSO service provider partner, this parameter is required. It is the identifier for the group of custom properties that are associated with the SSO service provider partner. This parameter is specified as an integer.
-idpId	This parameter is optional. It is the IdP identifier for the group of custom properties that are to be defined with this command. If the parameter is not specified, an unused identifier is assigned. This parameter is specified as an integer.

Table 66. importSAMLIdpMetaData parameters (continued)

Parameter	Description
-signingCertAlias	This parameter is optional if you do not have a signing certificate. If you have a signing certificate, this parameter is required. This parameter specifies the alias that you want the certificate to be named in the current keystore. This parameter is specified as a Boolean.
-idpMetadataFileName	This parameter is required. Specify the fully-qualified file name for the SAML IdP partner metadata. This parameter is specified as a String.
-securityDomainName	This parameter specifies the name of the security domain of interest. If a value for this parameter is not specified, the command uses the global security configuration. This parameter is specified as a String.

Results

The IdP partner properties are now added to the SAML TAI for this WebSphere Application Server.

Example

The following example imports the SAML IdP partner 1 metadata to the global security SAML TAI SSO service provider partner 1 with a signing certificate alias name idp1CertAlias:

```
AdminTask.importSAMLIdpMetadata('-idpMetadataFileName /tmp/myIdPmetadata.xml
                                 -ssoId 1 -idpId 1 -signingCertAlias idp1CertAlias')
```

The following example imports the SAML IdP partner 1 metadata to the security domain myDomain1 SAML TAI SSO service provider partner 1 with a signing certificate alias name idp1CertAlias:

```
AdminTask.iportSAMLIdpMetadata('-idpMetadataFileName /tmp/myIdPmetadata.xml
                               -ssoId 1 -idpId 1 -signingCertAlias idp1CertAlias -securityDomainName myDomain1')
```

Displaying SAML identity provider (IdP) partner configuration using the wsadmin command-line utility

About this task

You can use the wsadmin command-line utility to display the Security Assertion Markup Language (SAML) trust association interceptor (TAI) identity provider (IdP) partner configuration in the security configuration for WebSphere Application Server.

Procedure

1. Start the WebSphere Application Server.
2. Start the wsadmin command utility from the app_server_root/bin directory by entering the command: wsadmin -lang jython.
3. At the wsadmin prompt, enter the following command:

   ```
   AdminTask.showSAMLIdpPartner('-ssoId 1')
   ```

   You can use the following parameters with this command:

Table 67. showSAMLIdpPartner parameters

Parameter	Description
`-ssoId`	This parameter is optional if you have only one SSO service provider partner. If you have more than one SSO service provider partner, this parameter is required. It is the identifier for the group of custom properties that are associated with the SSO service provider partner. This parameter is specified as an integer.
`-idpId`	This parameter specifies the identifier of the IdP whose properties you want to display. If a value for this parameter is not specified, the command shows all IdP partners for the specified SSO service provider partner. This parameter is specified as an integer.
`-securityDomainName`	This parameter specifies the name of the security domain of interest. If a value for this parameter is not specified, the command uses the global security configuration. This parameter is specified as a String.

Results

The custom properties for the specified SAML web SSO IdP partner are displayed.

Example

The following example displays the SAML Idp partner 1 of the SSO service provider partner 1 from the global security SAML TAI configuration:

```
AdminTask.showSAMLIdpPartner('-ssoId 1 -idpId 1')
```

The following example displays the SAML IdP partner 2 of the SSO service provider partner 1 from the security domain `myDomain1`:

```
AdminTask.showSAMLIdpPartner('-ssoId 1 -idpId 2 -securityDomainName myDomain1')
```

Displaying SAML web single sign-on (SSO) trust association interceptor (TAI) configuration using the wsadmin command-line utility

About this task

You can use the `wsadmin` command-line utility to display the Security Assertion Markup Language (SAML) web single sign-in (SSO) trust association interceptor (TAI) in the security configuration for WebSphere Application Server.

Procedure

1. Start the WebSphere Application Server.
2. Start the `wsadmin` command utility from the `app_server_root/bin` directory by entering the command: `wsadmin -lang jython`.
3. At the `wsadmin` prompt, enter the following command:

   ```
   AdminTask.showSAMLTAISSO()
   ```

   You can use the following parameters with this command:

Table 68. showSAMLTAISSO parameters

Parameter	Description
`-ssoId`	This parameter specifies an SSO service provider partner identifier for which the TAI properties need to be displayed. If this parameter is not specified, all SSO partners are displayed. This parameter is specified as an integer.
`-securityDomainName`	This parameter specifies the name of the security domain of interest. If a value for this parameter is not specified, the command uses the global security configuration. This parameter is specified as a String.

Results

The SAML TAI custom properties for this WebSphere Application Server are displayed.

Example

The following example displays the SAML TAI custom properties of the SSO service provider partner 1 from the global security configuration:

```
AdminTask.showSAMLTAISSO('-ssoId 1)
```

The following example displays the SAML TAI custom properties of the SSO service provider partner 1 from the security domain `myDomain1`:

```
AdminTask.showSAMLTAISSO('-ssoId 1 -securityDomainName myDomain1')
```

Chapter 8. Authorizing access to resources

WebSphere Application Server provides many different methods for authorizing accessing resources. For example, you can assign roles to users and configure a built-in or external authorization provider.

About this task

You can create an application, an Enterprise JavaBeans (EJB) module, or a web module and secure them using assembly tools.

To authorize user or group access to resources, read the following articles:

Procedure

1. Secure you application during assembly and deployment. For more information on how to create a secure application using an assembly tool, such as the IBM Rational Application Developer, see the information about securing applications during assembly and deployment.
2. Authorize access to Java Platform, Enterprise Edition (Java EE) resources. WebSphere Application Server supports authorization that is based on the Java Authorization Contract for Containers (JACC) specification in addition to the default authorization. When security is enabled in WebSphere Application Server, the default authorization is used unless a JACC provider is specified. For more information, see “Authorization providers” on page 578.
3. Authorize access to administrative resources. You can assign users and groups to predefined administrative roles such as the monitor, configurator, operator, administrator, auditor and iscadmins roles. These roles determine which tasks a user can perform in the administrative console. For more information, see “Authorizing access to administrative roles” on page 632.

What to do next

After authorizing access to resources, configure the Application Server for secure communication. For more information, see Chapter 9, “Securing communications,” on page 667.

Authorization technology

Authorization information determines whether a user or group has the necessary privileges to access resources.

WebSphere Application Server supports many authorization technologies including the following:

- Authorization involving the web container and Java Platform, Enterprise Edition (Java EE) technology
- Authorization involving an enterprise bean application and Java EE technology
- Authorization involving web services and Java EE technology
- Java Message Service (JMS)
- Java Authorization Contract for Containers (JACC)

  WebSphere Application Server supports both a default authorization provider and an authorization provider that is based on the Java Authorization Contract for Containers (JACC) specification. The JACC-based authorization provider enables third-party security providers to handle the Java EE authorization. For more information, see “JACC support in WebSphere Application Server” on page 579.
- Java Authentication and Authorization Service (JAAS)

  For more information, see “Java Authentication and Authorization Service” on page 437.
- Java 2 security

  For more information, see “Java 2 security” on page 74.
- Naming and administrative authorization

- Pluggable authorization

  WebSphere Application Server supports an authorization infrastructure that enables you to plug in an external authorization provider. For more information, see “Enabling an external JACC provider” on page 602.

Administrative roles and naming service authorization

WebSphere Application Server extends the Java Platform, Enterprise Edition (Java EE) security role-based access control to protect the product administrative and naming subsystems.

Administrative roles

A number of administrative roles are defined to provide the degrees of authority that are needed to perform certain WebSphere Application Server administrative functions from either the administrative console or the system management scripting interface called wsadmin. The authorization policy is only enforced when administrative security is enabled. The following table describes the administrative roles:

Table 69. Administrative roles that are available through the administrative console and wsadmin.

This table lists administrative roles that are available through the administrative console and wsadmin.

Role	Description
Monitor	An individual or group that uses the monitor role has the least amount of privileges. A monitor can complete the following tasks: • View the WebSphere Application Server configuration. • View the current state of the Application Server.
Configurator	An individual or group that uses the configurator role has the monitor privilege plus the ability to change the WebSphere Application Server configuration. The configurator can perform all the day-to-day configuration tasks. For example, a configurator can complete the following tasks: • Create a resource. • Map an application server. • Install and uninstall an application. • Deploy an application. • Assign users and groups-to-role mapping for applications. • Set up Java 2 security permissions for applications. • Customize the Common Secure Interoperability Version 2 (CSIv2), Secure Authentication Service (SAS), and Secure Sockets Layer (SSL) configurations. **Important:** SAS is supported only between Version 6.0.x and previous version servers that have been federated in a Version 6.1 cell.
Operator	An individual or group that uses the operator role has monitor privileges plus ability to change the runtime state. For example, an operator can complete the following tasks: • Stop and start the server. • Monitor the server status in the administrative console.

Table 69. Administrative roles that are available through the administrative console and wsadmin (continued).

This table lists administrative roles that are available through the administrative console and wsadmin.

Role	Description
Administrator	An individual or group that uses the administrator role has the operator and configurator privileges plus additional privileges that are granted solely to the administrator role. For example, an administrator can complete the following tasks: • Modify the server user ID and password. • Configure authentication and authorization mechanisms. • Enable or disable administrative security. **Note:** In previous releases of WebSphere Application Server, the **Enable administrative security** option is known as the **Enable global security** option. • Enforce Java 2 security using the **Use Java 2 security to restrict application access to local resources** option. • Change the Lightweight Third Party Authentication (LTPA) password and generate keys. • Create, update, or delete users in the federated repositories configuration. • Create, update, or delete groups in the federated repositories configuration. **Note:** An administrator cannot map users and groups to the administrator roles. For information on how to assign federated repository management rights to users who are not assigned the WebSphere Application Server Administrator role, see the topic, *Mapping users and groups to roles for assigning federated repository management rights* in the topic, *Providing security*.
Adminsecuritymanager	Only users who are granted this role can map users to administrative roles. Also, when fine-grained administrative security is used, only users who are granted this role can manage authorization groups. See “Administrative roles” on page 574 for more information.
Deployer	Users who are granted this role can perform both configuration actions and runtime operations on applications.
Auditor	Users granted this role can view and modify the configuration settings for the security auditing subsystem. For example, a user with the auditor role can complete the following tasks: • Enable and disable the security auditing subsystem. • Select the event factory implementation to be used with the event factory plug-in point. • Select and configure the service provide, or emitter. or both to be used with the service provider plug-in point. • Set the audit policy that describes the behavior of the application server in the event of an error with the security auditing subsystem. • Define which security events are to be audited. The auditor role includes the monitor role. This allows the auditor to view but not change the rest of the security configuration.

Table 70. Additional administrative role that is available through the administrative console.

This table lists an additional administrative role that is available through the administrative console.

Role	Description
iscadmins	This role is only available for administrative console users and not for wsadmin users. Users who are granted this role have administrator privileges for managing users and groups in the federated respositories. For example, a user of the iscadmins role can complete the following tasks: • Create, update, or delete users in the federated repositories configuration. • Create, update, or delete groups in the federated repositories configuration.

Table 71. Additional administrative role that is available through wsadmin.

This table lists an additional administrative role that is available through the administrative console.

Role	Description
Deployer	This role is only available for wsadmin users and not for administrative console users. Users who are granted this role can perform both configuration actions and run-time operations on applications.

When administrative security is enabled, the administrative subsystem role-based access control is enforced. The administrative subsystem includes the security server, the administrative console, the wsadmin scripting tool, and all the Java Management Extensions (JMX) MBeans. When administrative security is enabled, both the administrative console and the administrative scripting tool require users to provide the required authentication data. Moreover, the administrative console is designed so the control functions that display on the pages are adjusted, according to the security roles that a user has. For example, a user who has only the monitor role can see only the non-sensitive configuration data. A user with the operator role can change the system state.

When you are changing registries (for example, from a federated repository to LDAP), make sure you remove the information that pertains to the previously configured registry for console users and console groups.

When administrative security is enabled, WebSphere Application Servers run under the server identity that is defined under the active user registry configuration. Although it is not shown on the administrative console and in other tools, a special Server subject is mapped to the administrator role. The WebSphere Application Server runtime code, which runs under the server identity, requires authorization to runtime operations. If no other user is assigned administrative roles, you can log into the administrative console or to the wsadmin scripting tool using the server identity to perform administrative operations and to assign other users or groups to administrative roles. Because the server identity is assigned to the administrative role by default, the administrative security policy requires the administrative role to perform the following operations:

- Change server ID and server password
- Enable or disable WebSphere Application Serveradministrative security
- Enforce Java 2 security using the **Use Java 2 security to restrict application access to local resources** option.
- Change the LTPA password or generate keys
- Assign users and groups to administrative roles

Version 6.1 release of WebSphere Application Server and subsequent releases require the following:

- An administrative user, distinguished from the server user identity, to improve auditability of administrative actions. The user name specifies a user with administrative privileges that is defined in the local operating system.
- Distinguish the server identity from the administrative user identity to improve auditability. The server user identity is used for authenticating server-to-server communications.
- The internal server ID enables the automatic generation of the user identity for server-to-server authentication. Automatic generation of the server identity supports improved auditability for cells only for Version 6.1 or later nodes. In the Version 6.1 release of WebSphere Application Server, you can

save the internally-generated server ID because the Security WebSphere Common Configuration Model (WCCM) model contains a new tag, internalServerId. You do not need to specify a server user ID and a password during security configuration except in a mixed-cell environment. An internally-generated server ID adds a further level of protection to the server environment because the server password is not exposed as it is in releases prior to Version 6.1. However, to maintain backwards compatibility, you must specify the server user ID if you use earlier versions of WebSphere Application Server.

When enabling security, you can assign one or more users and groups to naming roles. For more information, see Assigning users to naming roles. However, before assigning users to naming roles, configure the active user registry. User and group validation depends on the active user registry. For more information, see Configuring user registries.

- Ability to map a *special-subject* to the administrative roles. A special-subject is a generalization of a particular class of users. The AllAuthenticated or the AllAuhenticatedInTrustedRealms (when cross realm is involved) special subjects mean that the access check of the administrative role ensures that the user making the request is at least authenticated. The Everyone special subject means that anyone, authenticated or not, can perform the action as if security is not enabled.

Naming service authorization

CosNaming security offers increased granularity of security control over CosNaming functions. CosNaming functions are available on CosNaming servers such as the WebSphere Application Server. These functions affect the content of the WebSphere Application Server name space. Generally, you have two ways in which client programs result in CosNaming calls. The first is through the Java Naming and Directory Interface (JNDI) call. The second is with common object request broker architecture (CORBA) clients invoking CosNaming methods directly.

Four security roles are introduced :

- CosNamingRead
- CosNamingWrite
- CosNamingCreate
- CosNamingDelete

The roles have authority levels from low to high:

CosNamingRead
: You can query the WebSphere Application Server name space, using, for example, the JNDI lookup method. The special-subject, Everyone, is the default policy for this role.

CosNamingWrite
: You can perform write operations such as JNDI bind, rebind, or unbind, and CosNamingRead operations. As a default policy, Subjects are not assigned this role.

CosNamingCreate
: You can create new objects in the name space through such operations as JNDI createSubcontext and CosNamingWrite operations. As a default policy, Subjects are not assigned this role.

CosNamingDelete
: You can destroy objects in the name space, for example using the JNDI destroySubcontext method and CosNamingCreate operations. As a default policy, Subjects are not assigned this role.

A Server special-subject is assigned to all of the four CosNaming roles by default. The Server special-subject provides a WebSphere Application Server process, which runs under the server identity, to access all the CosNaming operations. The Server special-subject does not display and cannot be modified through the administrative console or other administrative tools.

Special configuration is not required to enable the server identity as specified when enabling administrative security for administrative use because the server identity is automatically mapped to the administrator role.

Users, groups, or the special subjects AllAuthenticated and Everyone can be added or removed to or from the naming roles from the WebSphere Application Server administrative console at any time. However, a server restart is required for the changes to take effect.

A best practice is to map groups or one of the special-subjects, rather than specific users, to naming roles because it is more flexible and easier to administer in the long run. By mapping a group to a naming role, adding or removing users to or from the group occurs outside of WebSphere Application Server and does not require a server restart for the change to take effect.

The CosNaming authorization policy is only enforced when administrative security is enabled. When administrative security is enabled, attempts to do CosNaming operations without the proper role assignment result in an org.omg.CORBA.NO_PERMISSION exception from the CosNaming server.

Each CosNaming function is assigned to only one role. Therefore, users who are assigned the CosNamingCreate role cannot query the name space unless they have also been assigned CosNamingRead. And in most cases a creator needs to be assigned three roles: CosNamingRead, CosNamingWrite, and CosNamingCreate. The CosNamingRead and CosNamingWrite roles assignment for the creator example are included in the CosNamingCreate role. In most of the cases, WebSphere Application Server administrators do not have to change the roles assignment for every user or group when they move to this release from a previous one.

Although the ability exists to greatly restrict access to the name space by changing the default policy, unexpected org.omg.CORBA.NO_PERMISSION exceptions can occur at runtime. Typically, Java EE applications access the name space and the identity they use is that of the user that authenticated to WebSphere Application Server when accessing the Java EE application. Unless the Java EE application provider clearly communicates the expected naming roles, use caution when changing the default naming authorization policy.

Administrative roles for business level applications

The Java 2 Platform, Enterprise Edition (J2EE) role-based authorization concept is extended to protect the WebSphere Application Server administrative subsystem. This protection applies to those administrative roles associated with business level applications.

Deploying business level applications on a server configured to hold business level applications requires a number of administrative roles that are defined to provide degrees of authority when performing certain administrative functions from either the Web-based administrative console or the system management scripting interface. The authorization policy is only enforced when administrative security is enabled. The following table describes the system management scripting command used for business level applications and the corresponding administrative role that is required in using the command:

Table 72. Business level application - administrative roles. Business level application - administrative roles

Command	Role Required
startBLA	Cell deployer, Cell operator, BLA deployer, BLA operator, Target deployer, Target operator
stopBLA	Cell deployer, Cell operator, BLA deployer, BLA operator, Target deployer, Target operator
createEmptyBLA	Cell configurator, Cell deployer
editBLA	Cell configurator, Cell deployer, BLA deployer
viewBLA	Cell monitor, BLA monitor
listBLAs	Cell monitor, BLA monitor(s)
deleteBLA	Cell configurator, Cell deployer, BLA developer
importAsset	Cell configurator, Cell deployer
editAsset	Cell configurator, Cell deployer, Asset deployer

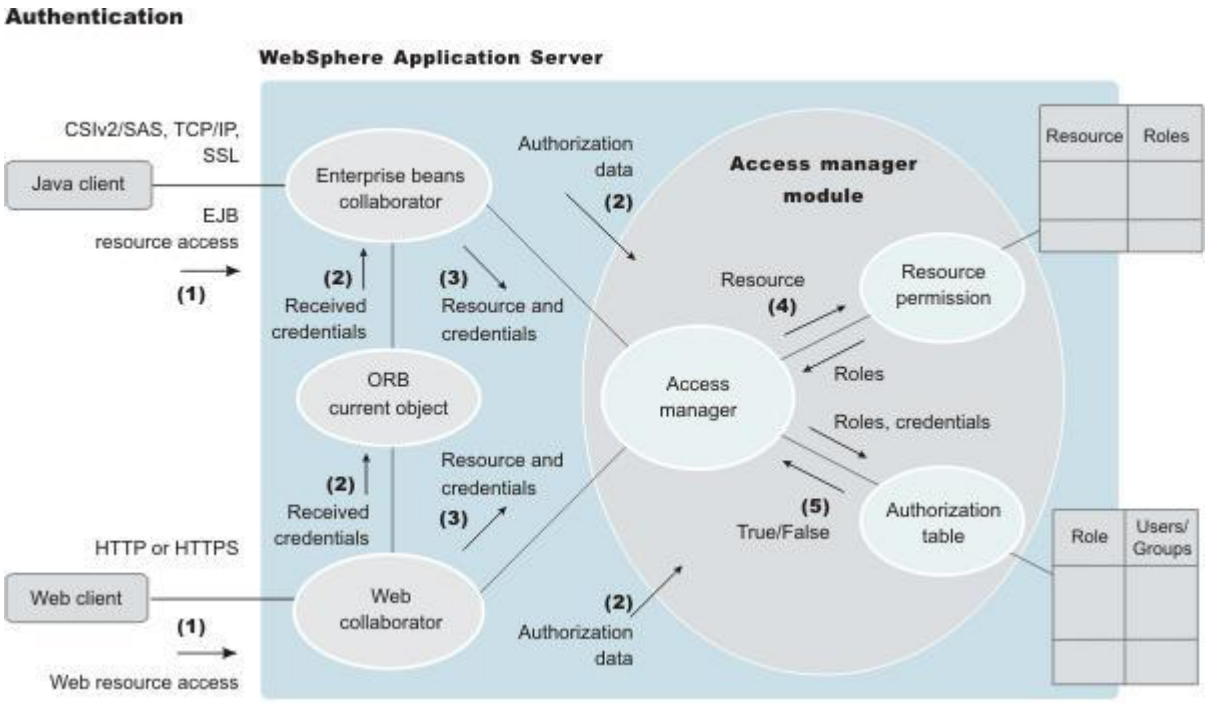
Table 72. Business level application - administrative roles (continued). Business level application - administrative roles

Command	Role Required
viewAsset	Cell monitor, Asset monitor(s)
listAssets	Cell monitor, Asset monitor
exportAsset	Cell monitor, Asset monitor
deleteAsset	Cell configurator, Cell deployer, Asset deployer
updateAsset	Cell configurator, Cell deployer, Asset deployer
addCompUnit	Cell configurator, Cell deployer, BLA deployer *(for the BLA to add the composition unit)* + Asset-deployer *(for the asset to create the composition unit from)* + Target-deployer *(for each target the composition unit is deployed to)* + Relationship-deployer *(for each relationship the composition unit depends on that will result in creating a composition unit from the dependency asset)*
editCompUnit	Cell configurator, Cell deployer, BLA deployer *(for the BLA this composition unit belongs to)* + Target deployer *(for each target that this composition unit is deployed to)*
viewCompUnit	Cell monitor, BLA monitor
listCompUnit	Cell monitor, BLA monitor
deleteCompUnit	Cell configurator, Cell deployer, BLA deployer *(for the BLA this composition unit belongs to)* + Target deployer *(for each target that this composition unit is deployed to)*
setCompUnitTargetAutoStart	Cell configurator, Cell deployer
listControlOps	Cell monitor, BLA monitor
getBLAStatus	Cell monitor, BLA monitor
	Where: • BLA deployer specifies the deployer role for the BLA that is being managed. • BLA monitor specifies the monitor role for the BLA that is being managed. • BLA operator specifies the operator role for the BLA that is being managed. • Asset deployer specifies the deployer role for the asset that is being managed. • Asset monitor specifies the monitor role for the asset that is being managed. • Target deployer specifies the deployer for the target that the composition unit is being deployed to. • Target operator specifies the operator role for the target that the composition unit is being deployed to.

Role-based authorization

Use authorization information to determine whether a caller has the necessary privileges to request a service.

The following figure illustrates the process that is used during authorization.



Web resource access from a web client is handled by a web collaborator. The Enterprise JavaBeans (EJB) resource access from a Java client, whether an enterprise bean or a servlet, is handled by an EJB collaborator. The EJB collaborator and the web collaborator extract the client credentials from the object request broker (ORB) current object. The client credentials are set during the authentication process as received credentials in the ORB current object. The resource and the received credentials are presented to the WSAccessManager access manager to check whether access is permitted to the client for accessing the requested resource.

The access manager module contains two main modules:

- The resource permission module helps determine the required roles for a given resource. This module uses a resource-to-roles mapping table that is built by the security runtime during application startup. To build the resource-to-role mapping table, the security runtime reads the deployment descriptor of the enterprise beans or the Web module (`ejb-jar.xml` file or `web.xml` file)
- The authorization table module consults a role-to-user or group table to determine whether a client is granted one of the required roles. The role-to-user or group mapping table, also known as the authorization table, is created by the security runtime during application startup.

  To build the authorization table, the security run time reads the application binding file, the `ibm-application-bnd.xmi` file, or the `ibm-application-bnd.xml` file, as appropriate.

  Note: For IBM extension and binding files, the .xmi or .xml file name extension is different depending on whether you are using a pre-Java EE 5 application or module or a Java EE 5 or later application or module. An IBM extension or binding file is named ibm-*-ext.xmi or ibm-*-bnd.xmi where * is the type of extension or binding file such as app, application, ejb-jar, or web. The following conditions apply:

  – For an application or module that uses a Java EE version prior to version 5, the file extension must be .xmi.

  – For an application or module that uses Java EE 5 or later, the file extension must be .xml. If .xmi files are included with the application or module, the product ignores the .xmi files.

However, a Java EE 5 or later module can exist within an application that includes pre-Java EE 5 files and uses the .xmi file name extension.

The `ibm-webservices-ext.xmi`, `ibm-webservices-bnd.xmi`, `ibm-webservicesclient-bnd.xmi`, `ibm-webservicesclient-ext.xmi`, and `ibm-portlet-ext.xmi` files continue to use the .xmi file extensions.

Use authorization information to determine whether a caller has the necessary privilege to request a service. You can store authorization information many ways. For example, with each resource, you can store an access-control list, which contains a list of users and user privileges. Another way to store the information is to associate a list of resources and the corresponding privileges with each user. This list is called a capability list.

WebSphere Application Server uses the Java 2 Platform, Enterprise Edition (J2EE) authorization model. In this model, authorization information is organized as follows:

During the assembly of an application, permission to invoke methods is granted to one or more roles. A role is a set of permissions; for example, in a banking application, roles can include teller, supervisor, clerk, and other industry-related positions. The teller role is associated with permissions to run methods that are related to managing the money in an account, such as the withdraw and deposit methods. The teller role is not granted permission to close accounts; this permission is given to the supervisor role. The application assembler defines a list of method permissions for each role. This list is stored in the deployment descriptor for the application.

Three special subjects are not defined by the J2EE model: AllAuthenticatedUsers, AllAuthenticatedInTrustedRealms, and Everyone. A special subject is a product-defined entity that is defined outside of the user registry. This entity is used to generically represent a class of users or groups in the registry.

- The AllAuthenticatedUsers subject permits all authenticated users to access protected methods. As long as the user can authenticate successfully, the user is permitted access to the protected resource.
- The AllAuthenticatedInTrustedRealms subject permits all authenticated foreign users (those that are bound to other realms) to access protected methods. As long as the user can authenticate successfully, the user is permitted access to the protected resource.
- The Everyone subject permits unrestricted access to a protected resource. Users do not have to authenticate to get access; this special subject provides access to protected methods as if the resources are unprotected.

During the deployment of an application, real users or groups of users are assigned to the roles. When a user is assigned to a role, the user gets all the method permissions that are granted to that role.

The application deployer does not need to understand the individual methods. By assigning roles to methods, the application assembler simplifies the job of the application deployer. Instead of working with a set of methods, the deployer works with the roles, which represent semantic groupings of the methods.

Users can be assigned to more than one role; the permissions that are granted to the user are the union of the permissions granted to each role. Additionally, if the authentication mechanism supports the grouping of users, these groups can be assigned to roles. Assigning a group to a role has the same effect as assigning each individual user to the role.

A best practice during deployment is to assign groups instead of individual users to roles for the following reasons:

- Improves performance during the authorization check. Typically far fewer groups exist than users.
- Provides greater flexibility, by using group membership to control resource access.

- Supports the addition and deletion of users from groups outside of the product environment. This action is preferred to adding and removing them to WebSphere Application Server roles. Stop and restart the enterprise application for these changes to take effect. This action can be very disruptive in a production environment.

At runtime, WebSphere Application Server authorizes incoming requests based on the user's identification information and the mapping of the user to roles. If the user belongs to any role that has permission to run a method, the request is authorized. If the user does not belong to any role that has permission, the request is denied.

The J2EE approach represents a declarative approach to authorization, but it also recognizes that you cannot deal with all situations declaratively. For these situations, methods are provided for determining user and role information programmatically. For enterprise beans, the following two methods are supported by WebSphere Application Server:
- **`getCallerPrincipal`**: This method retrieves the user identification information.
- **`isCallerInRole`**: This method checks the user identification information against a specific role.

For servlets, the following methods are supported by WebSphere Application Server:
- getRemoteUser
- isUserInRole
- getUserPrincipal

These methods correspond in purpose to the enterprise bean methods.

For more information on the J2EE security authorization model, see the following website:
http://java.sun.com

Administrative roles

The Java Platform, Enterprise Edition (Java EE) role-based authorization concept is extended to protect the WebSphere Application Server administrative subsystem.

A number of administrative roles are defined to provide degrees of authority that are needed to perform certain administrative functions from either the Web-based administrative console or the system management scripting interface. The authorization policy is only enforced when administrative security is enabled. The following table describes the administrative roles:

Table 73. Administrative roles. Administrative roles

Role	Description
Monitor	An individual or group that uses the monitor role has the least amount of privileges. A monitor can complete the following tasks: • View the WebSphere Application Server configuration. • View the current state of the Application Server.

Table 73. Administrative roles (continued). Administrative roles

Role	Description
Configurator	An individual or group that uses the configurator role has the monitor privilege plus the ability to change the WebSphere Application Server configuration. The configurator can perform all the daily configuration tasks. For example, a configurator can complete the following tasks: • Create a resource. • Map an application server. • Install and uninstall an application. • Deploy an application. • Assign users and groups-to-role mapping for applications. • Set up Java 2 security permissions for applications. • Customize the Common Secure Interoperability Version 2 (CSIv2), Security Authentication Service (SAS), and Secure Sockets Layer (SSL) configurations. **Important:** SAS is supported only between Version 6.0.x and previous version servers that have been federated in a Version 6.1 cell.
Operator	An individual or group that uses the operator role has monitor privileges plus ability to change the runtime state. For example, an operator can complete the following tasks: • Stop and start the server. • Monitor the server status in the administrative console.
Administrator	An individual or group that uses the administrator role has the operator and configurator privileges, plus additional privileges that are granted solely to the administrator role. For example, an administrator can complete the following tasks: • Modify the server user ID and password. • Configure authentication and authorization mechanisms. • Enable or disable administrative security. • Enable or disable Java 2 security. • Change the Lightweight Third Party Authentication (LTPA) password and generate keys. • Create, update, or delete users in the federated repositories configuration. • Create, update, or delete groups in the federated repositories configuration. **Important:** An administrator cannot map users and groups to the administrator roles without also having the adminsecuritymanager role.
iscadmins	This role is only available for administrative console users, not for wsadmin users. Users who are granted this role have administrator privileges for managing users and groups in the federated repositories. For example, a user of the iscadmins role can complete the following tasks: • Create, update, or delete users in the federated repositories configuration. • Create, update, or delete groups in the federated repositories configuration.
Deployer	Users granted this role can complete both configuration actions and runtime operations on applications. See the “Deployer role” on page 576 section for more details.
Admin Security Manager	You can assign users and groups to the Admin Security Manager role on the cell level through wsadmin scripts and the administrative console. Using the Admin Security Manager role, you can assign users and groups to the administrative user roles and administrative group roles. However, an administrator cannot assign users and groups to the administrative user roles and administrative group roles including the Admin Security Manager role. See the “Admin Security Manager role” on page 577 section for more details.

Table 73. Administrative roles (continued). Administrative roles

Role	Description
Auditor	Users granted this role can view and modify the configuration settings for the security auditing subsystem. For example, a user with the auditor role can complete the following tasks: • Enable and disable the security auditing subsystem. • Select the event factory implementation to be used with the event factory plug-in point. • Select and configure the service provide, or emitter. or both to be used with the service provider plug-in point. • Set the audit policy that describes the behavior of the application server in the event of an error with the security auditing subsystem. • Define which security events are to be audited. The auditor role includes the monitor role. This allows the auditor to view but not change the rest of the security configuration. See the "Auditor role" on page 578 section for more details.

The server ID that is specified and the administrative ID, if specified, when enabling administrative security is automatically mapped to the administrator role.

Users and groups can be added or removed from administrative roles using the WebSphere Application Server administrative console by a user given the appropriate authority. The Primary administrative user name must be used to log on to the administrative console to change the administrative user and group roles other than the auditor role. Only a user with the auditor role can change the auditor user and group roles. When security auditing is initially enabled, the Primary administrative user is also given the auditor role, and can manage all of the administrative user and group roles including the those in the auditor role. A best practice is to map a group or groups, rather than specific users, to administrative roles because it is more flexible and easier to administer.

In addition to mapping user or groups, a special-subject can also be mapped to the administrative roles. A special-subject subject is a generalization of a particular class of users. The AllAuthenticated special subject means that the access check of the administrative role ensures that the user making the request is at least authenticated. The Everyone special subject means that anyone, authenticated or not, can perform the action, as if security was not enabled.

Deployer role

A user that is granted a deployer role can complete all of the configuration and runtime operations on an application. A deployer role can be subsets of both configurator and operator roles. However, a user granted a deployer role cannot configure or operate any other resources, such as a server, node.

When fine-grained administrative security is used, only a user granted a deployer role to an application can configure and operate that application.

Cell-level configurators can configure applications. Cell-level operators can also operate (start and stop) applications. However, a user granted a deployer role at cell level can also complete configuration and operation on all applications.

Table 74. Deployer role capabilities.

This table lists the deployer role capabilities when fine-grained administrative security is used.

Operation	Required Roles (Any one)
Install application	Cell-configurator, target-deployer
Uninstall application	Cell-configurator, application-deployer, target-deployer

Table 74. Deployer role capabilities (continued).

This table lists the deployer role capabilities when fine-grained administrative security is used.

Operation	Required Roles (Any one)
List application	Cell-monitor, application-monitor
Edit, update and redeploy application	Cell-configurator, application-deployer
Export application	Cell-monitor, application-monitor
Start or stop application	Cell-operator, application-deployer

Where:

Cell-configurator
Specifies the configurator role at cell level.

Application-deployer
Specifies the deployer role for the application that is being managed.

Target-deployer
Specifies the deployer role for all servers or clusters for which an application is targeted. If you have a target-deployer role, you can install a new application on the target. However, to edit or update the installed application, you must be included in the authorization group of the installed application-deployer.

The target-deployer cannot explicitly start or stop a new application. However, when a target-deployer starts a server on a target, all of the applications that have their auto-start attribute set to `yes` are started when the server starts.

It is recommended that the application-deployer set this attribute to `true` if the application-deployer does not want the application to be started by the target-deployer.

Admin Security Manager role

The Admin Security Manager role separates administrative security administration from other application administration.

By default, serverId and adminID, if specified, are assigned to this role in the cell level authorization table. This role implies a monitor role. However, an administrator role does not imply the Admin Security Manager role.

When fine-grained admin security is used, only a user granted this role at cell level can manage administrative authorization groups. However, a user granted this role for each administrative authorization group can map users to administrative roles for those groups. The following list summarizes the capabilities of the Admin Security Manager role at different levels, such as the cell and administrative authorization group levels.

Table 75. Admin Security Manager role capabilities.

This table lists Admin Security Manager role capabilities.

Action	Role
Map users to administrative roles for cell level	Only the Admin Security Manager of the cell
Map users to administrative roles for an authorization group	Only the Admin Security Manager of that authorization group or the Admin Security Manager of the cell
Manage authorization groups, create, delete, add resource to an authorization group, or remove resource from an authorization group or list	Only the Admin Security Manager of the cell

Auditor role

The auditor role separates security auditing administration from administrative security and other application administration.

The auditor role was added to allow distinct separation of the authority of an auditor from the authority of the administrator. The auditor role can be granted to administrators to combine their authority. When security is first enabled, the auditor role is assigned to the primary administrator. If in your situation the separation of authority is required, administrators can remove the auditor role from themselves and assign the auditor role to other users.

A fine grained security for the auditor role is not implemented, which results in the auditor role requiring the monitor role. This process allows the auditor to read but not modify the panels managed by the administrator. The auditor has full authority to read and modify the panels associated with the security auditing subsystem. The administrator will have the monitor role for those panels, however, the administrator cannot modify those panels.

Authorization providers

WebSphere Application Server supports authorization that is based on the Java Authorization Contract for Containers (JACC) specification in addition to the default authorization.

JACC is a specification introduced in Java Platform, Enterprise Edition (Java EE)1.4. It enables third-party security providers to manage authorization in the application server.

Note: In WebSphere Application Server version 7.0, Java Authorization Contract for Containers (JACC) specification 1.4 was applied. In JACC specification 1.4, we support Java EE5 that includes Servlet 2.5 and EJB 3. The biggest functional change with the introduction of JACC specification 1.4 is the inclusion of annotations for propagating security policy information.

When security is enabled in WebSphere Application Server, the default authorization is used unless a JACC provider is specified. The default authorization does not require special setup, and the default authorization engine makes all of the authorization decisions. However, if a JACC provider is configured and set up for WebSphere Application Server to use, all of the enterprise beans and web authorization decisions are delegated to the JACC provider.

WebSphere Application Server supports security for Java EE applications and also for its administrative components. Java EE applications, such as Web and Enterprise JavaBeans (EJB) components are protected and authorized per the Java EE specification. The administrative components are internal to WebSphere Application Server and are protected by the role-based authorizer. The administrative components include the administrative console, MBeans, and other components such as naming and security. For more information on administrative security, see “Role-based authorization” on page 571.

When a JACC provider is used for authorization in WebSphere Application Server, all of the Java EE application-based authorization decisions are delegated to the provider per the JACC specification. However, all administrative security authorization decisions are made by the WebSphere Application Server default authorization engine. The JACC provider is not called to make the authorization decisions for administrative security.

When a protected Java EE resource is accessed, the authorization decision to give access to the principal is the same whether using the default authorization engine or a JACC provider. Both of the authorization models satisfy the J2EE specification, and function the same. Choose a JACC provider only when you want to work with an external security provider such as Tivoli Access Manager. In this instance, the security provider must support the JACC specification and be set up to work with WebSphere Application

Server. Setting up and configuring a JACC provider requires additional configuration steps, depending on the provider. Unless you have an external security provider that you can use with WebSphere Application Server, use the default authorization.

JACC support in WebSphere Application Server

WebSphere Application Server supports the Java Authorization Contract for Containers (JACC) specification, which enables third-party security providers to handle the Java Platform, Enterprise Edition (Java EE) authorization.

The JACC specification requires that both the containers in the application server and the provider satisfy some requirements. Specifically, the containers are required to propagate the security policy information to the provider during the application deployment and to call the provider for all authorization decisions. The providers are required to store the policy information in their repository during application deployment. The providers then use this information to make authorization decisions when called by the container.

JACC access decisions

When security is enabled and an enterprise bean or web resource is accessed, the Enterprise JavaBeans (EJB) container or web container calls the security runtime to make an authorization decision on whether to permit access. When using an external provider, the access decision is delegated to that provider.

According to the Java Authorization Contract for Containers (JACC) specification, the appropriate permission object is created, the appropriate policy context handlers are registered, and the appropriate policy context identifier (contextID) is set. A call is made to the java.security.Policy object method that is implemented by the provider to make the access decision.

The following sections describe how the provider is called for both the enterprise bean and the web resources.

Access decisions for enterprise beans

When security is enabled, and an EJB method is accessed, the EJB container delegates the authorization check to the security runtime. If JACC is enabled, the security runtime uses the following process to perform the authorization check:

1. Creates the EJBMethodPermission object using the bean name, method name, interface name, and the method signature.
2. Creates the context ID and sets it on the thread by using the PolicyContext.setContextID(contextID) method.
3. Registers the required policy context handlers, including the Subject policy context handler.
4. Creates the ProtectionDomain object with principal in the Subject. If no principal exists, null is passed for the principal name.
5. The access decision is delegated to the JACC provider by calling the implies method of the Policy object, which is implemented by the provider. The EJBMethodPermission and the ProtectionDomain objects are passed to this method.
6. The isCallerInRole access check also follows the same process, except that an EJBRoleRefPermission object is created instead of an EJBMethodPermission object.

Access decisions for web resources

When security is enabled and configured to use a JACC provider, and when a web resource such as a servlet or a JavaServer Pages (JSP) file is accessed, the security runtime delegates the authorization decision to the JACC provider by using the following process:

1. A WebResourcePermission object is created to see if the URI is cleared. If the provider honors the Everyone subject it is also selected here.

a. The WebResourcePermission object is constructed with the urlPattern and the HTTP method accessed.
b. A ProtectionDomain object with a null principal name is created.
c. The JACC provider Policy.implies method is called with the permission and the protection domain. If the URI access is cleared or given access to the Everyone subject, the provider permits access (return `true`) in the implies method. Access is then granted without further checks.

2. If access is not granted in the previous step, a WebUserDataPermission object is created and used to see if the Uniform Resource Identifier (URI) is precluded, excluded or must be redirected using the HTTPS protocol.
   a. The WebUserDataPermission object is constructed with the urlPattern accessed, the HTTP method invoked, and the transport type of the request. If the request is over HTTPS, the transport type is set to CONFIDENTIAL; otherwise, null is passed.
   b. A ProtectionDomain object with a null principal name is created.
   c. The JACC provider Policy.implies method is called with the permission and the protection domain. If the request is using the HTTPS protocol and the implies method returns `false`, the HTTP 403 error is returned to imply excluded and precluded permission. In this case no further checks are performed. If the request is not using the HTTPS protocol, and the implies returns `false`, the request is redirected over HTTPS.
3. The security runtime attempts to authenticate the user. If the authentication information already exists (for example, LTPA token), it is used. Otherwise, the user's credentials must be entered.
4. After the user credentials are validated, a final authorization check is performed to see if the user is granted access privileges to the URI.
   a. As in Step 1, the WebResourcePermission object is created. The ProtectionDomain object now contains the Principal that is attempting to access the URI. The Subject policy context handler also contains the user's information, which can be used for the access check.
   b. The provider implies method is called using the Permission object and the ProtectionDomain object created previously. If the user is granted permission to access the resource, the implies method returns `true`. If the user is not granted access, the implies method returns `false`.

Even if the order listed previously is changed later (for example, to improve performance) the end result is the same. For example, if the resource is precluded or excluded, the end result is that the resource cannot be accessed.

For more information on these access permissions, see the JSR-000115 Java Authorization Contract for Containers (Final Release).

Using information from the Subject for access decision

If the provider relies on the WebSphere Application Server generated Subject for access decision, the provider can query the public credentials in the Subject to obtain the WSCredential credential. The WSCredential API is used to obtain information about the user, including the name and the groups that the user belongs to. This information is used to make the access decision.

If the provider adds the required information to the Subject, WebSphere Application Server can use the information to make the access decision. The provider might add the information by using the Trust Association Interface feature or by plugging login modules into the Application Server.

The security attribute propagation section contains additional documentation on how to add the WebSphere Application Server required information to the Subject. For more information, see “Propagating security attributes among application servers” on page 473.

Dynamic module updates in JACC

WebSphere Application Server supports dynamic updates to web modules under certain conditions. If a web module is updated, deleted or added to an application, only that module is stopped and started as appropriate. The other existing modules in the application are not impacted, and the application itself is not stopped and then restarted.

When using the default authorization engine, any security policies are modified in the web modules and the application is stopped and then restarted. When using the Java Authorization Contract for Containers (JACC) based authorization, the behavior depends on the functionality that a provider supports. If a provider can handle dynamic changes to the web modules, then only the web modules are impacted. Otherwise, the entire application is stopped and restarted for the new changes in the web modules to take effect.

A provider can indicate if it supports the dynamic updates by configuring the `Supports dynamic module updates` option in the JACC configuration model (see “Authorizing access to Java EE resources using Tivoli Access Manager” on page 597 for more information). This option can be enabled or disabled using the administrative console or by scripting. It is expected that most providers store the policy information in their external repository, which makes it possible for them to support these dynamic updates. This option should be enabled by default for most providers.

When the `Supports dynamic module updates` option is enabled, if a web module that contains security roles is dynamically added, modified, or deleted, only the specific web modules are impacted and restarted. If the option is disabled, the entire application is restarted. When dynamic updates are performed, the security policy information of the modules impacted are propagated to the provider. For more information about security policy propagation, see “JACC policy propagation” on page 583.

Initialization of the JACC provider

If a Java Authorization Contract for Containers (JACC) provider requires initialization during server startup, for example, to enable the client code to communicate to the server code, the provider can implement the com.ibm.wsspi.security.authorization.InitializeJACCProvider interface. See “Interfaces that support JACC” on page 616 for more information.

When this interface is implemented, it is called during server startup. Any custom properties in the JACC configuration model are propagated to the initialize method of this implementation. The custom properties can be entered using either the administrative console or by scripting.

During server shutdown, the cleanup method is called for any clean-up work that a provider requires. Implementation of this interface is strictly optional, and is used only if the provider requires initialization during server startup.

Mixed node environment and JACC

Authorization using Java Authorization Contract for Containers (JACC) is a new feature in WebSphere Application Server Version 6.0.x. Also, the JACC configuration is set up at the cell level and is applicable for all the nodes and servers in that cell.

If you are planning to use the JACC-based authorization, the cell must contain Version 6.0.x and later nodes only. This restriction implies that a mixed node environment containing a set of Version 5.x nodes in a Version 6.0.x or later cell is not supported.

JACC providers

The Java Authorization Contract for Containers (JACC) is a specification that was first introduced in Java Platform, Enterprise Edition (Java EE) Version 1.4 through the Java Specifications Request (JSR) 115

process. JACC specification 1.4 is included for WebSphere Application Server version 7.0 for Java EE 5 support.. This specification defines a contract between Java EE 5 containers and authorization providers.

The contract enables third-party authorization providers to plug into Java EE 5 application servers, such as WebSphere Application Server, to make the authorization decisions when a Java EE 5 resource is accessed. The access decisions are made through the standard java.security.Policy object.

To plug in to WebSphere Application Server, the third-party JACC provider must implement the policy class, policy configuration factory class, and policy configuration interface, which are all required by the JACC specification.

The JACC specification does not specify how to handle the authorization table information between the container and the provider. It is the responsibility of the provider to provide some management facilities to handle this information. The container is not required to provide the authorization table information in the binding file to the provider.

WebSphere Application Server provides the RoleConfigurationFactory and the RoleConfiguration role configuration interfaces to help the provider obtain information from the binding file, as well as an initialization interface (InitializeJACCProvider). The implementation of these interfaces is optional. See “Interfaces that support JACC” on page 616 for more information about these interfaces.

Tivoli Access Manager as the default JACC provider for WebSphere Application Server

The JACC provider in WebSphere Application Server is implemented by both the client and the server pieces of the Tivoli Access Manager. The client piece of Tivoli Access Manager is embedded in WebSphere Application Server. The server piece is located on a separate installable CD that is shipped as part of the WebSphere Application Server, Network Deployment (ND) package.

The JACC provider is not the default authorization. You must configure WebSphere Application Server to use the JACC provider.

JACC policy context handlers

WebSphere Application Server supports all of the policy context handlers that are required by the Java Authorization Contract for Containers (JACC) specification. However, due to performance impacts, the Enterprise JavaBeans (EJB) arguments policy context handler is not activated unless it is specifically required by the provider. Performance impacts result if objects must be created for each argument of each EJB method.

If the provider supports and requires this context handler, select the **Requires the EJB arguments policy context handler for access decisions** check box in the External JACC provider link under the Authorization providers panel or by using scripting. Any changes to this option are effective after the servers are restarted. By default this option is disabled. Disable this option when using Tivoli Access Manager as the JACC provider, because the argument values are not required for access decisions.

JACC policy context identifiers (ContextID) format

A policy context identifier is defined as a unique string that represents a policy context. A policy context contains all of the security policy statements as defined by the Java Contract for Containers (JACC) specification that affect access to the resources in a web or Enterprise JavaBeans (EJB) module.

During policy propagation to the JACC provider, a PolicyConfiguration object is created for each policy context. The object is populated with the policy statements, represented by the JACC permission objects that correspond to the context. The object is propagated to the JACC provider using the JACC specification APIs.

Note: The following information is include for planning purposes and is only applicable if you intend to federate in the future.

WebSphere Application Server makes the contextID unique by using the `href:cellName/appName/moduleName` string as the contextID format for the modules. The `href` part of the string indicates that a hierarchical name is passed as the context ID. The `cellName` represents the name of the deployment manager cell or the base cell where the application is installed.

The `appName` part of the string in the context ID represents the application name containing the module. The `moduleName` refers to the name of the module.

As an example, the context ID for the module `Increment.jar` file in an application named DefaultApplication that is installed in `cell1` is the `href:cell1/DefaultApplication/Increment.jar` file.

JACC policy propagation

When an application is installed or deployed in WebSphere Application Server, the security policy information in the application is propagated to the provider when the configuration is saved. The context ID for the application is saved in its `application.xml` file, that is used for propagating the policy to the Java Authorization Contract for Containers (JACC) provider, and also for access decisions for Java Platform, Enterprise Edition (Java EE) resources.

Note: This topic references one or more of the application server log files. As a recommended alternative, you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log` , `SystemErr.log`, `trace.log`, and `activity.log` files on distributed and IBM i systems. You can also use HPEL in conjunction with your native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

When an application is uninstalled, the security policy information in the application is removed from the provider when the configuration is saved.

If the provider implemented the RoleConfiguration interface, the security policy information that is propagated to the policy provider also contains the authorization table information. See “Interfaces that support JACC” on page 616 for more information about this interface.

If an application does not contain security policy information, the PolicyConfiguration (and the RoleConfiguration, if implemented) objects do not contain any information. The existence of empty PolicyConfiguration and RoleConfiguration objects indicates that security policy information for the module does not exist.

After an application is installed, it can be updated without being uninstalled and reinstalled. For example, a new module can be added to an existing application, or an existing module can be modified. In this instance, the information in the impacted modules is propagated to the provider by default. A module is impacted when the deployment descriptor of the module or annotations within the module are changed as part of the update. If the provider supports the RoleConfiguration interfaces, the entire authorization table for that application is propagated to the provider.

If the security information is not propagated to the provider during application updates, you can set the com.ibm.websphere.security.jacc.propagateonappupdate Java virtual machine (JVM) property to `false` in the deployment manager, in a Network Deployment environment, or the unmanaged base application server. If this property is set to `false`, any updates to an existing application in the server are not propagated to the provider. You also can set this property on a per-application basis using the custom properties of an application. The wsadmin tool can be used to set the custom property of an application. If this property is set at the application level, any updates to that application are not propagated to the provider. If the update to an application is a full update, for example, a new application enterprise archive (EAR) file is used to replace the existing one, and the provider is refreshed with the entire application security policy information.

As mentioned earlier, the security policy information is propagated to the JACC provider during the save operation. The `SystemOut.log` file indicates the success or failure of the propagation to the provider. Check the log file after the installation to ensure that the propagation had no problems. If the propagation had any problems, access to the application fails when Tivoli Access Manager is used as the JACC provider.

If the security policy information for the application is successfully propagated to the provider, the audit statements with the message key SECJ0415I appear. However, if there was a problem propagating the security policy information to the provider (for example: network problems, JACC provider is not available), the `SystemOut.log` files contain the error message with the message keys SECJ0396E during install or SECJ0398E during modification. The installation of the application is not stopped due to a failure to propagate the security policy to the JACC provider. Also, in the case of failure, no exception or error messages appear during the save operation. When the problem causing this failure is fixed, run the **propagatePolicyToJaccProvider** tool to propagate the security policy information to the provider without reinstalling the application. For more information, see Propagating security policy of installed applications to a JACC provider using wsadmin scripting.

JACC registration of the provider implementation classes

The JACC specification states that providers can plug in their provider using the javax.security.jacc.policy.provider and the javax.security.jacc.PolicyConfigurationFactory.provider system properties.

The javax.security.jacc.policy.provider property is used to set the policy object of the provider, while the javax.security.jacc.PolicyConfigurationFactory.provider property is used to set the provider PolicyConfigurationFactory implementation.

Although both system properties are supported in WebSphere Application Server, it is highly recommended that you use the configuration model that is provided. You can set these values using either the JACC configuration panel (see “Authorizing access to Java EE resources using Tivoli Access Manager” on page 597 for more information) or by using wsadmin scripting. One of the advantages of using the configuration model instead of the system properties is that the information is entered in one place at the cell level, and is propagated to all nodes during synchronization. Also, as part of the configuration model, additional properties can be entered, as described in the JACC configuration panel.

Role-based security with embedded Tivoli Access Manager

The Java Platform, Enterprise Edition (Java EE) role-based authorization model uses the concepts of roles and resources. An example is provided here.

Table 76. Roles.

This table is an example of role-based security with embedded Tivoli Access Manager.

Roles	getBalance	deposit	closeAccount
Teller	granted	granted	
Cashier	granted		
Supervisor			granted

In the example of the banking application that is conceptualized in the previous table, three roles are defined: teller, cashier, and supervisor. Permission to perform the getBalance, deposit, and closeAccount application methods are mapped to these roles. From the example, you can see that users assigned the role, Supervisor, can run the closeAccount method, whereas the other two roles are unable to run this method.

The term, principal, within WebSphere Application Sever security refers to a person or a process that performs activities. *Groups* are logical collections of principals that are configured in WebSphere Application Server to promote the ease of applying security. Roles can be mapped to principals, groups, or both.

Table 77. Roles methods. The entry that is invoked in the following table indicates that the principal or group can invoke any methods that are granted to that role.

Principal/Group	Teller	Cashier	Supervisor
TellerGroup	Invoke		
CashierGroup		Invoke	
SupervisorGroup			
Frank: A principal who is not a member of any of the previous groups		Invoke	Invoke

In the previous example, the principal Frank, can invoke the getBalance and the closeAccount methods, but cannot invoke the deposit method because this method is not granted either the Cashier or the Supervisor role.

At the time of application deployment, the Java Authorization Contract for Container (JACC) provider of Tivoli Access Manager populates the Tivoli Access Manager-protected object space with any security policy information that is contained in the application deployment descriptor and or annotations. This security information is used to determine access whenever the WebSphere Application Server resource is requested.

By default, the Tivoli Access Manager access check is performed using the role name, the cell name, the application name, and the module name.

Tivoli Access Manager access control lists (ACLs) determine which application roles are assigned to a principal. ACLs are attached to the applications in the Tivoli Access Manager-protected object space at the time of application deployment.

Principal-to-role mappings are managed from the WebSphere Application Server administrative console and are never modified using Tivoli Access Manager. Direct updates to ACLs are performed for administrative security users only.

The following sequence of events occur:

1. During application deployment, policy information is sent to the JACC provider of Tivoli Access Manager . This policy information contains permission-to-role mappings and role-to-principal and role-to-group mapping information.
2. The JACC provider of Tivoli Access Manager converts the information into the required format, and passes this information to the Tivoli Access Manager policy server.
3. The policy server adds entries to the Tivoli Access Manager-protected object space to represent the roles that are defined for the application and the permission-to-role mappings. A permission is represented as a Tivoli Access Manager-protected object and the role that is granted to this object is attached as an extended attribute.

Tivoli Access Manager integration as the JACC provider

Tivoli Access Manager uses the Java Authorization Contract for Container (JACC) model in WebSphere Application Server to perform access checks.

Tivoli Access Manager consists of the following components:

- Run time
- Client configuration
- Authorization table support
- Access check
- Authentication using the PDLoginModule module

For the run-time changes, Tivoli Access Manager implements the PolicyConfigurationFactory and the PolicyConfiguration interfaces, as required by JACC. During the application installation, the security policy

information in the deployment descriptor and the authorization table information in the binding files are propagated to the Tivoli provider using these interfaces. The Tivoli provider stores the policy and the authorization table information in the Tivoli Access Manager policy server by calling the respective Tivoli Access Manager application programming interfaces (API).

Tivoli Access Manager also implements the RoleConfigurationFactory and the RoleConfiguration interfaces. These interfaces are used to ensure that the authorization table information is passed to the provider with the policy information. See “Interfaces that support JACC” on page 616 for more information about these interfaces.

To configure the Tivoli Access Manager client, you can use either the administrative console or wsadmin scripting. You can access the administrative console panels for the Tivoli Access Manager client configuration by clicking **Security > Global security > External authorization providers**. Under Related Items, click **External JACC provider**. The Tivoli client must be set up to use the Tivoli Access Manager JACC Provider.

For more information about how to configure the Tivoli Access Manager client, see “Tivoli Access Manager JACC provider configuration” on page 605.

Tivoli Access Manager uses the RoleConfiguration interface to ensure that the authorization table information is passed to the Tivoli Access Manager provider when the application is installed or deployed. When an application is deployed or edited, the set of users and groups for the user or group-to-role mapping are obtained from the Tivoli Access Manager server, which shares the same Lightweight Directory Access Protocol (LDAP) server as WebSphere Application Server. This sharing is accomplished by plugging into the application management users or groups-to-role administrative console panels. The management APIs are called to obtain users and groups rather than relying on the WebSphere Application Server-configured LDAP registry.

When WebSphere Application Server is configured to use the JACC provider for Tivoli Access Manager , it passes the information to Tivoli Access Manager to make the access decision. The Tivoli Access Manager policy implementation queries the local replica of the access control list (ACL) database for the access decision.

The custom login module in WebSphere Application Server can do the authentication. This login module is plugged in before the WebSphere Application Server-provided login modules. The custom login modules can provide information that can be stored in the Subject. If the required information is stored, no additional registry calls are made to obtain that information.

As part of the JACC integration, the Tivoli Access Manager-provided PDLoginModule module is also used to plug into WebSphere Application Server for Lightweight Third Party Authentication (LTPA), Kerberos (KRB5) and Simple WebSphere Authentication Mechanism (SWAM) authentication. The PDLoginModule module is modified to authenticate with the user ID or password. The module is also used to fill in the required attributes in the Subject so that no registry calls are made by the login modules in WebSphere Application Server. The information that is placed in the Subject is available for the Tivoli Access Manager policy object to use for access checking.

Note: SWAM is deprecated in WebSphere Application Server Version 8.5 and will be removed in a future release.

Note: When using Kerberos authentication mechanism and Tivoli Access Manager, TAM loginModule creates the PDPrincipal without first going through the Tivoli Access Manager authentication process. Also when using Kerberos authentication mechanism and Tivoli Access Manager, the Tivoli Access Manager policy is not enforced starting in WebSphere Application Server Version 7.0.

Tivoli Access Manager security for WebSphere Application Server

WebSphere Application Server provides embedded IBM Tivoli Access Manager client technology to secure your WebSphere Application Server-managed resources.

The benefits of using Tivoli Access Manager that are described here are only applicable when Tivoli Access Manager client code is used with the Tivoli Access Manager server:

- Robust container-based authorization
- Centralized policy management
- Management of common identities, user profiles, and authorization mechanisms
- Single-point security management for Java Platform, Enterprise Edition (Java EE) compliant and non-compliant Java EE resources using the administrative console for Tivoli Access Manager Web Portal Manager
- No requirements for coding or deployment changes to applications
- Easy management of users, groups, and roles using the WebSphere Application Server administrative console

WebSphere Application Server supports the Java Authorization Contract for Containers (JACC) specification. JACC details the contract requirements for Java EE containers and authorization providers. With this contract, authorization providers can perform the access decisions for resources in Java EE application servers such as WebSphere Application Server. The Tivoli Access Manager security utility that is embedded within WebSphere Application Server is JACC-compliant and is used to:

- Add security policy information when applications are deployed
- Authorize access to WebSphere Application Server-secured resources.

When applications are deployed, the embedded Tivoli Access Manager client takes any policy and or user and role information that is stored within the application deployment descriptor or using annotations and stores it within the Tivoli Access Manager Policy Server.

The Tivoli Access Manager JACC provider is also called when a user requests access to a resource that is managed by WebSphere Application Server.

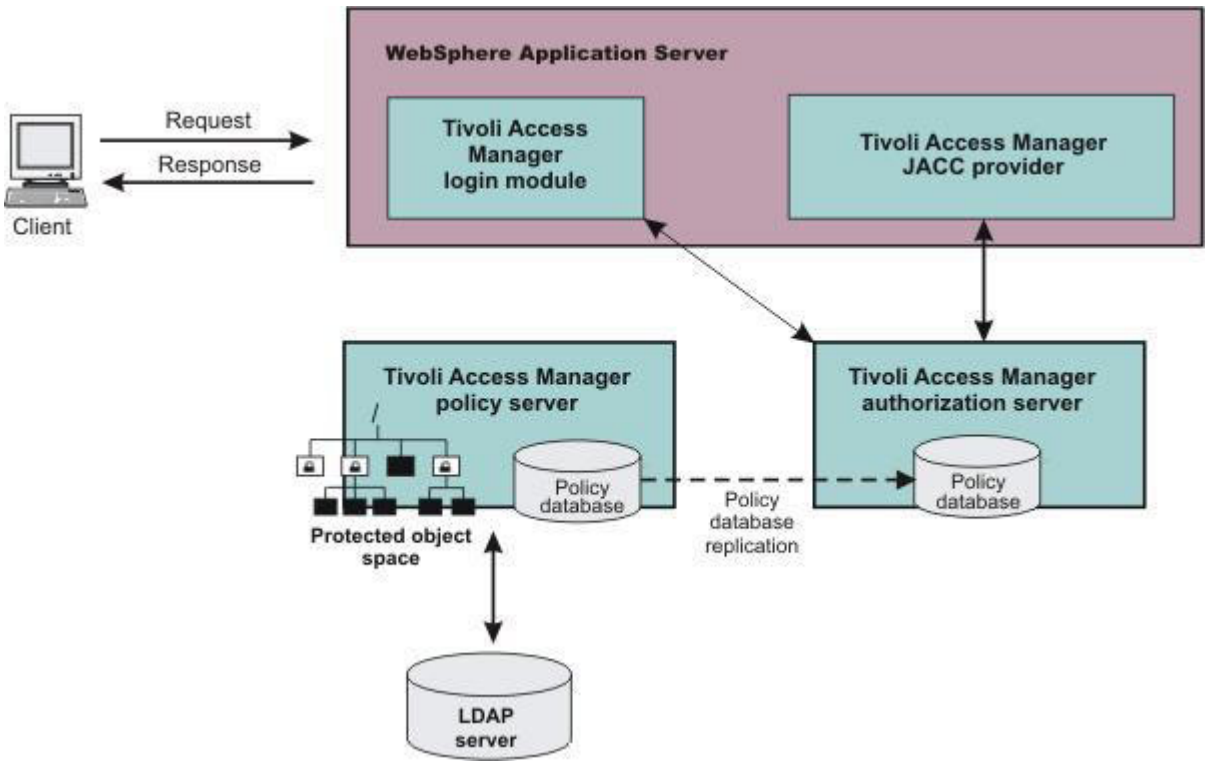


Figure 33. Embedded Tivoli Access Manager client architecture

The previous figure illustrates the following sequence of events:

1. Users that access protected resources are authenticated using the Tivoli Access Manager login module that is configured for use when the embedded Tivoli Access Manager client is enabled.
2. The WebSphere Application Server container uses information from the Java EE application deployment descriptor and annotations to determine the required role membership.
3. WebSphere Application Server uses the embedded Tivoli Access Manager client to request an authorization decision from the Tivoli Access Manager authorization server. Additional context information, when present, is also passed to the authorization server. This context information is comprised of the cell name, Java EE application name, and Java EE module name. If the Tivoli Access Manager policy database has policies that are specified for any of the context information, the authorization server uses this information to make the authorization decision.
4. The authorization server consults the permissions that are defined for the specified user within the Tivoli Access Manager-protected object space. The protected object space is part of the policy database.
5. The Tivoli Access Manager authorization server returns the access decision to the embedded Tivoli Access Manager client.
6. WebSphere Application Server either grants or denies access to the protected method or resource, based on the decision that is returned from the Tivoli Access Manager authorization server.

At its core, Tivoli Access Manager provides an authentication and authorization framework. You can learn more about Tivoli Access Manager, including the information that is necessary to make deployment decisions, by reviewing the product documentation. The following guides are available in the IBM Tivoli Access Manager for e-business Information Center:

- *IBM Tivoli Access Manager for e-business Installation Guide*

  This guide describes how to plan, install, and configure a Tivoli Access Manager secure domain. Using a series of easy installation scripts, you can quickly deploy a fully functional secure domain. These scripts are very useful when prototyping the deployment of a secure domain.

To access this guide in the IBM Tivoli Access Manager for e-business information center, click **Access Manager for e-business > Installation and upgrade information > Installation Guide**.

- *IBM Tivoli Access Manager for e-business Administration Guide*

  This document presents an overview of the Tivoli Access Manager security model for managing protected resources. This guide describes how to configure the Tivoli Access Manager servers that make access control decisions. In addition, detailed instructions describe how to perform important tasks, such as declaring security policies, defining protected object spaces, and administering user and group profiles.

  To access this guide in the IBM Tivoli Access Manager for e-business information center, click **Access Manager for e-business >Administration Information > Administration Guide**.

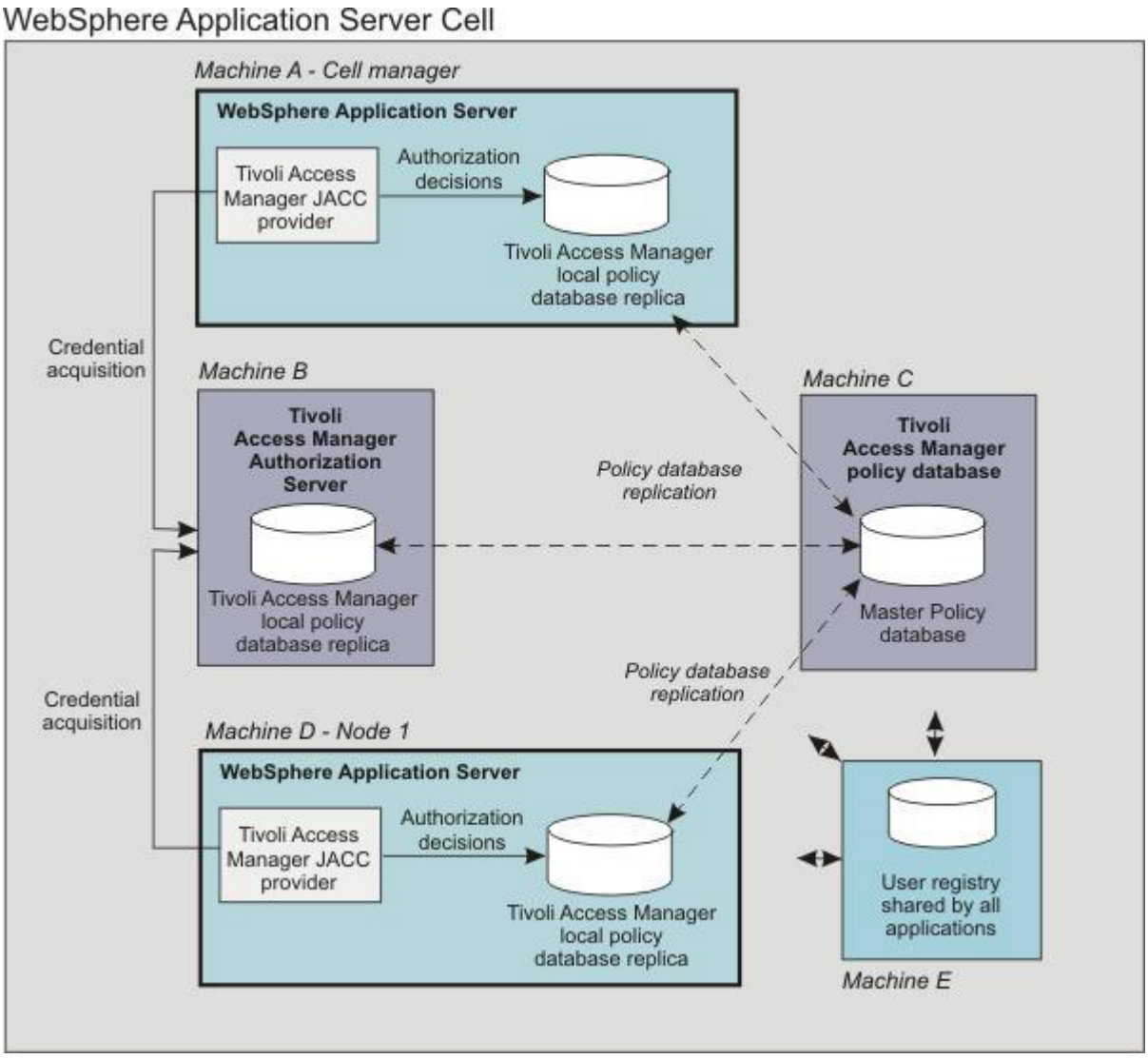


Figure 34. Tivoli Access Manager provides centralized administration of multiple servers

The previous figure is an example architecture showing WebSphere Application Servers secured by Tivoli Access Manager.

The participating WebSphere Application Servers use a local replica of the Tivoli Access Manager policy database to make authorization decisions for incoming requests. The local policy databases are replicas of the master policy database. The master policy database is installed as part of the Tivoli Access Manager

installation. Having policy database replicas on each participating WebSphere Application Server node optimizes performance when making authorization decisions and provides failover capability.

Although the authorization server can also be installed on the same system as WebSphere Application Server, this configuration is not illustrated in the diagram.

All instances of Tivoli Access Manager and WebSphere Application Server in the example architecture share the Lightweight Directory Access Protocol (LDAP) user registry on Machine E.

The LDAP registries that are supported by WebSphere Application Server are also supported by Tivoli Access Manager.

It is possible to have separate WebSphere Application Server profiles on the same host that is configured for different Tivoli Access Manager servers. Such an architecture requires that the profiles are configured for separate Java SE Runtime Environments (JRE 6) and therefore you need multiple JREs installed on the same host.

Security annotations

Annotations are a powerful programming mechanism resulting from the JSR-175 recommendation. An annotation is a standard way to include supported security behaviors while allowing, the source code and configuration files to be generated automatically.

In Java Platform, Enterprise Edition (Java EE) 5 and later, The security roles and policies can be defined using annotations as well as within the deployment descriptor. During the installation of the application, the security policies and roles defined using annotations are merged with the security policies and roles defined within the deployment descriptor. This merge is performed by the Annotations Metadata Manager (AMM) facility. When the metadata is merged, the following inheritance rules are followed.

Table 78. Metadata merger inheritance rules.

This table lists the metadata merger inheritance rules.

Scenario	Rules
Security metadata in deployment descriptor only	No merge is needed, the security metadata from the deployment descriptor is propagated.
Security metadata in annotations only	No merge is needed, the security metadata defined with annotations is propagated.
Security metadata in deployment descriptor and annotations	The metadata from the deployment descriptor and annotations is merged. The metadata in annotations is overridden by the same type of data from the deployment descriptor.

Six security annotations are currently supported. For each annotation, a MergeAction implementation is defined.

- @DeclareRoles (Servlet 2.5 and greater and EJB 3)

  The MergeAction implementation finds all the classes annotated with the DeclareRoles annotation. Within each annotated class for each role name specified, if the security roles listed in the deployment descriptor does not contain a SecurityRole with the annotated role name, a new SecurityRole is created and added to this list of security roles.
- @RunAs (Servlet 2.5 and greater and EJB 3)

  The MergeAction implementation finds all the classes with the RunAs annotation. For each annotated class, it finds the Servlet or the Enterprise Java Bean (EJB) associated with the given class. It then determines if a run-as element is defined in the deployment descriptor for the servlet or EJB. If one is not found, a new run-as element is created and added to the deployment descriptor. If a run-as element is found, this run-as element will be used instead of creating a new one. The role name used in the RunAs annotation must be defined in the deployment descriptor.
- @DenyAll (EJB 3 only)

The MergeAction implementation finds all the methods annotated with the DenyAll annotation. For each annotated method, if the method is not included in the deployment descriptor list of excluded methods, and a MethodPermission does not exist in the deployment descriptor, a new MethodElement is created and added to this list of excluded methods in the deployment descriptor.

- @PermitAll (EJB 3 only)

  The MergeAction implementation finds all the classes and the methods with the PermitAll annotation. For each annotated class, it finds the Enterprise Java Bean (EJB) associated with the given class. It then searches the subset of the MethodElements in the list of all the MethodPermissions defined in the deployment descriptor for this EJB. If a MethodElement with a wildcard method name ("*") is not found and a wildcard method does not exist in the list of excluded methods or in the list of MethodElements with security roles, a new MethodPermission and a new MethodElement are created. The new MethodPermission is marked unchecked and is added to the MethodPermission list in the deployment descriptor. The new MethodElement is added to the MethodElement list of the newly created unchecked MethodPermission. Similar action is done for all annotated methods. Instead of a wildcard MethodElement, the method signature must match exactly the signature of the annotated method.

- @RolesAllowed (EJB 3 only)

  The MergeAction implementation finds all of the classes and methods with the RolesAllowed annotation. For each annotated class, it finds the EJB associated with the given class. It then finds the subset of the MethodElements in the list of all the MethodPermissions defined in the deployment descriptor for this EJB. If a MethodElement with a wildcard method name ("*") is not found, and a wildcard method does not exist in the list of excluded methods or in the list of unchecked MethodElements, a new MethodPermission and MethodElement are created. If a MethodPermission for this EJB exists with exactly the same roles as those found in the annotation, this MethodPermission will be used instead of creating a new one. For each role name specified in the annotation, a new SecurityRole is created and added to the SecurityRole list in the MethodPermission, If the MethodPermission was newly created, it is added to the MethodPermission list in the deployment descriptor. The new MethodElement created is added to the MethodElement list of the MethodPermission. Similar processing is done for all annotated methods. Instead of a wildcard MethodElement, the method signature must exactly match the signature of the annotated method. Additionally, for each role name specified in the annotation, if the deployment descriptor list of security roles does not contain a SecurityRole with the annotated role name, a new SecurityRole is also created and added to this list of security roles.

- @ServletSecurity (Servlet 3.0 only)

  Note: Support for ServletSecurity annotation for Servlet 3.0 is new in this release of WebSphere Application Server.

  When an application deploys, the ServletSecurity MergeAction implementation finds all servlets with the ServletSecurity annotation. For each annotated servlet, it finds the servlet associated with the given class base on the WebServlet annotation. If RolesAllowed in the ServletSecurity annotation is not found in the deployment descriptor, it then creates a role-name attribute for the role in the deployment descriptor.

  When an application starts, the WebContainer inspects all servlets with the RunAs, declareRoles, and ServletSecurity annotations, and sets those annotations on the setServletSecurity() method of the ServletRegistration annotation. The WebContainer notifies the security component to inspect all ServletRegistration annotations that have URL patterns and security constraints. The security component then determines if a URL pattern is defined in the deployment descriptor. If one is not defined in the deployment descriptor, the security constraints and RunAs role in the URL pattern are created and then used. If an exact match is already defined in the deployment descriptor, the security constraints and RunAs role in the URL pattern of the deployment descriptor are used instead of the annotation data.

  Note: When the web authentication system property, com.ibm.wsspi.security.web.webAuthReq, is set to `persisting`, you can log into an unprotected URL if a valid username and password are provided.

The Inherited servlet annotation is a metadata annotation. Do not specify the Inherited annotation in the class. If a subclass does not have security annotation, it automatically inherits security annotation from the parent class. The subclass can overwrite the parent security annotations by specifying its security annotations.

The following example is for all HTTP methods with no constraints:

```
@WebServlet ("/Example")
@ServletSecurity
public class Example extends HttpServlet {
       ......
}
```

The following example is for all HTTP methods with no <auth-constraint> element and confidential TransportGuarantee required:

```
@WebServlet ("/Example")
@ServletSecurity(@HttpConstraint(transportGuarantee =
                     TransportGuarantee.CONFIDENTIAL))
public class Example extends HttpServlet {
       ......
}
```

The following example is for all HTTP methods with all access denied:

```
@WebServlet ("/Example")
@ServletSecurity(@HttpConstraint(EmptyRoleSemantic.DENY))
public class Example extends HttpServlet {
       ......
}
```

The following example is for all HTTP methods except for the GET and POST values with no constraints. For GET, the <auth-constraint> element requires membership in ALL ROLE. For POST, all access is denied.

```
@WebServlet (name="Example", urlPatterns={"/Example"})
@ServletSecurity((httpMethodConstraints = {
 @HttpMethodConstraint(value = "GET", rolesAllowed = "ALL ROLE"),
 @HttpMethodConstraint(value="POST", emptyRoleSemantic =
EmptyRoleSemantic.DENY))
})
public class Example extends HttpServlet {
       ......
}
```

The following example is for all HTTP methods except GET, the <auth-constraint> element requires membership in ALL ROLE, and the GET method has no constraints.

```
@WebServlet (name="Example", urlPatterns={"/Example"})
@ServletSecurity(value = @HttpConstraint(rolesAllowed = "ALL ROLE"),
 httpMethodConstraints = @HttpMethodConstraint("GET"))
public class Example extends HttpServlet {
       ......
}
```

The following example is for all HTTP methods except TRACE, the <auth-constraint> element requires membership in ALL ROLE, and for TRACE, all access is denied.

```
@WebServlet (name="Example", urlPatterns={"/Example"})
@ServletSecurity(value = @HttpConstraint(rolesAllowed = "ALL ROLE"),
 httpMethodConstraints = @HttpMethodConstraint(value="TRACE",
 emptyRoleSemantic = EmptyRoleSemantic.DENY))
public class Example extends HttpServlet {
       ......
}
```

Java Servlet 3.0 support for security

This release of WebSphere Application Server supports all security updates as defined in the Java Servlet 3.0 specification.

This release of WebSphere Application Server supports all security updates as defined in the Java Servlet 3.0 specification (JSR-315), including the new servlet security annotations, use of new programmatic security APIs and the dynamic updating of the servlet security configuration.

A significant enhancement is the new annotation support for servlets. A developer can declare the security constraints using annotations as an alternative to declaring them as part of the web.xml file, which is used prior to Java Servlet 3.0. The `web.xml` file continues to function and overrides any conflicts defined as annotations.

The list of supported Java Servlet 3.0 updates for security includes the following:

- Support for the @ServletSecurity annotation
- Support for the dynamic updating of the @RunAs, @declareRoles, and @ServletSecurity servlet security annotations
- Support for the authenticate, login and logout servlet security methods
- The new com.ibm.websphere.security.displayRealm property specifies whether the HTTP basic authentication login window displays the realm name that is not defined in the application `web.xml` file.

The following discusses the Java Servlet 3.0 updates for security in more detail:

Support for the @ServletSecurity annotation:

When an application deploys, the ServletSecurity MergeAction implementation finds all servlets with the ServletSecurity annotation. For each annotated servlet, it finds the servlet associated with the given class base on the WebServlet annotation. If RolesAllowed in the ServletSecurity annotation is not found in the deployment descriptor, it then creates a role-name attribute for the role in the deployment descriptor.

When an application starts, the WebContainer inspects all servlets with the RunAs, declareRoles, and ServletSecurity annotations, and sets those annotations on the setServletSecurity() method of the ServletRegistration annotation. The WebContainer notifies the security component to inspect all ServletRegistration annotations that have URL patterns and security constraints. The security component then determines if a URL pattern is defined in the deployment descriptor. If one is not defined in the deployment descriptor, the security constraints and RunAs role in the URL pattern are created and then used. If an exact match is already defined in the deployment descriptor, the security constraints and RunAs role in the URL pattern of the deployment descriptor are used instead of the annotation data.

Read the Security annotations topic for more information.

Support for the dynamic updating of the @RunAs, @declareRoles, and @ServletSecurity servlet security annotations:

When an application starts, the web container inspects all servlets with the RunAs, declareRoles, and ServletSecurity annotations, and sets those annotations on the setServletSecurity() method of the ServletRegistration annotation. The web container notifies the security component to inspect all ServletRegistration annotations that have URL patterns and security constraints. The security component then determines if a URL pattern is defined in the deployment descriptor. If an exact match is already defined in the deployment descriptor, the security constraints and RunAs role in the URL pattern of the deployment descriptor are used instead of the dynamic data.

Read the Servlet security dynamic annotations topic for more information.

Note: WebSphere Application Server supports both a default authorization provider and an authorization provider that is based on the Java Authorization Contract for Containers (JACC) specification. The JACC-based authorization provider (for example, the Tivoli Access Manager), enables third-party security providers to handle the Java EE authorization. The RunAs, declareRoles, and ServletSecurity annotations are supported for both native authorization and for JACC.

Support for the authenticate, login and logout servlet security methods:

The authenticate method authenticates a user by using the WebSphere Application Server container login mechanism configured for the servlet context.

The login method authenticates a user to the WebSphere Application Server with a user ID and password. If authentication is successful, it creates a user subject on the thread and Lightweight Third Party Authentication (LTPA) cookies (if single sign-on (SSO) is enabled).

The logout method logs the user out of the WebSphere Application Server and invalidates the HTTP session.

Read the Servlet security methods topic for more information.

The new `com.ibm.websphere.security.displayRealm` property specifies whether the HTTP basic authentication login window displays the realm name that is defined in the application `web.xml` file:

If the realm name is not defined in the `web.xml` file, one of the following occurs:

- If the property is set to `false` (the default), the WebSphere realm name display is Default Realm.
- If the property is set to `true`, the WebSphere realm name display is the user registry realm name for the LTPA authentication mechanism or the Kerberos realm name for the Kerberos authentication mechanism.

Read the Security custom properties topic for more information.

Servlet security dynamic annotations

When you use the programmatic APIs to add or to create a servlet, the security annotations, RunAs, declareRoles and ServletSecurity, can be dynamically updated through the setRunAsRole(), declareRoles() and setServletSecurity() methods respectively.

Note: Support for the dynamic updating of the RunAs, declareRoles, and ServletSecurity servlet security annotations is new in this release of WebSphere Application Server.

When an application starts, the web container inspects all servlets with the RunAs, declareRoles, and ServletSecurity annotations, and sets those annotations on the setServletSecurity() method of the ServletRegistration annotation. The web container notifies the security component to inspect all ServletRegistration annotations that have URL patterns and security constraints. The security component then determines if a URL pattern is defined in the deployment descriptor. If an exact match is already defined in the deployment descriptor, the security constraints and RunAs role in the URL pattern of the deployment descriptor are used instead of the dynamic data.

Note: If the dynamic security annotations, declareRoles, setRunAs and rolesAllowed, are used, the role name must be pre-defined, either through the deployment descriptor or through the declareRoles and or RunAs annotations in the servlet class. During deployment time, you can use the administrative console to map a user or group to this role.

If you have an exact URL pattern match for the ServletSecurity annotation in the security dynamic annotation, the security constraint of the URL pattern in the security dynamic annotation takes precedent. Also, if you call the setServletSecurity() method multiple times with the same URL pattern, the last one takes precedent.

- ServletRegistration.Dynamic.setRunAsRole(String roleName) sets the name of the RunAs role for this servlet registration.
- ServletContext.declareRoles(String roleNames) declares role names that are tested for the isUserInRole() method.
- ServletRegistration.Dynmaic.setServletSecurity(ServletSecurityElement constraint) sets the ServletSecurityElement for this servlet registration.

Note: When the web authentication system property, com.ibm.wsspi.security.web.webAuthReq, is set to `persisting`, you can log into an unprotected URL if a valid username and password are provided.

The following two examples can be used to set the security constraints and RunAs role for dynamic servlets by using the setServletSecurity() method.

In this example, all HTTP elements require membership in the Employee role except for the PUT method. For the PUT method, the <auth-constraint> element requires membership in the Manager role and TransportGuarantee is confidential.

```
HttpConstraintElement constraint = new HttpConstraintElement(TransportGuarantee.NONE,
new String[]{"Employee"});
List<HttpMethodConstraintElement> methodConstraints =
new ArrayList<HttpMethodConstraintElement>();
methodConstraints.add(new HttpMethodConstraintElement("PUT",
new HttpConstraintElement(TransportGuarantee.CONFIDENTIAL, new String[]{"Manager"})));
ServletSecurityElement servletSecurity =
new ServletSecurityElement(constraint, methodConstraints);
```

In this example, all HTTP methods are allowed except for the CUSTOM and GET methods. For the CUSTOM method, the <auth-constraint> element requires membership in the Manager role. For the GET method, the <auth-constraint> element requires membership in the Employee role, and TransportGuarantee is confidential.

```
HttpConstraintElement constraint = new HttpConstraintElement();
List<HttpMethodConstraintElement> methodConstraints =
new ArrayList<HttpMethodConstraintElement>();
methodConstraints.add(new HttpMethodConstraintElement("CUSTOM",
new HttpConstraintElement(TransportGuarantee.NONE, new String[]{"Manager"})));
methodConstraints.add(new HttpMethodConstraintElement("GET",
new HttpConstraintElement(TransportGuarantee.CONFIDENTIAL, new String[]{"Employee"})));
ServletSecurityElement servletSecurity = new ServletSecurityElement(constraint,
methodConstraints);
```

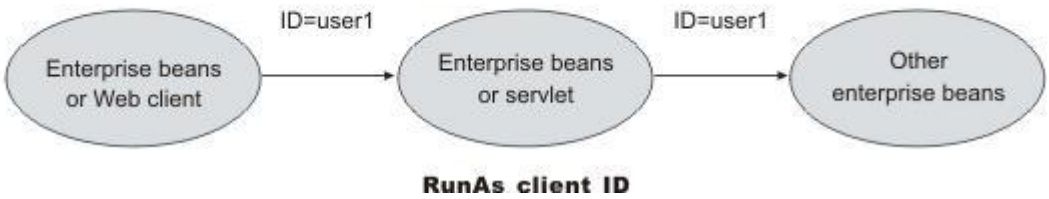
Delegations

Delegation is a process security identity propagation from a caller to a called object. As per the Java Platform, Enterprise Edition (Java EE) specification, a servlet and enterprise beans can propagate either the client or remote user identity when invoking enterprise beans, or they can use another specified identity as indicated in the corresponding deployment descriptor.

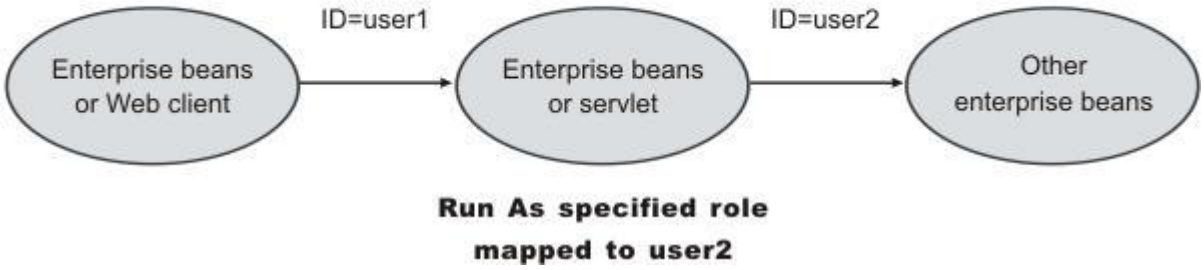
The extension supports enterprise bean propagation to the server ID when invoking other entity beans. Three types of delegations are possible:

- Delegate (RunAs) client identity
- Delegate (RunAs) specified identity
- Delegate (RunAs) system identity

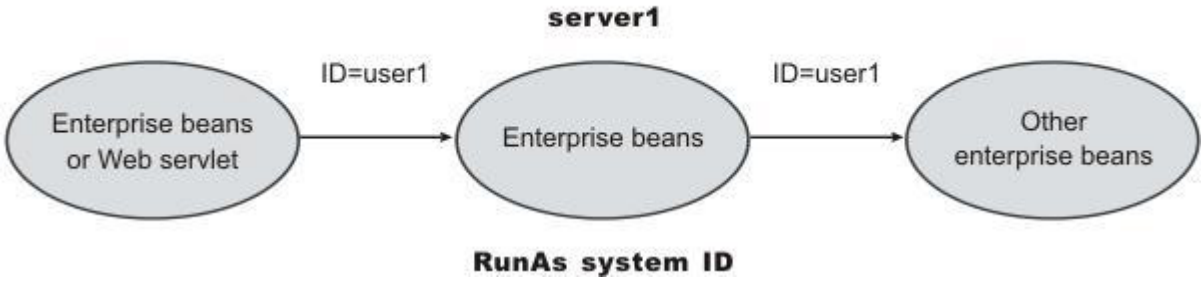
Delegate (RunAs) client identity



Delegate (RunAs) specified identity



Delegate (RunAs) system identity



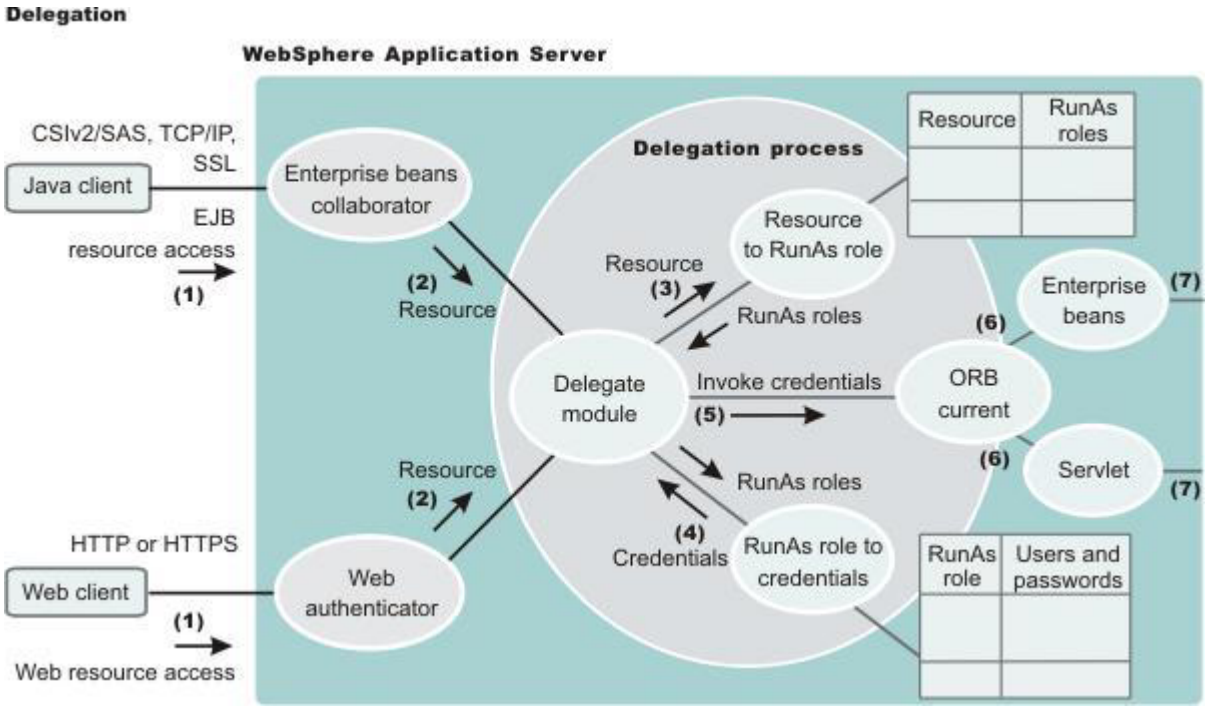
Note: The RunAs system identity delegation only works when server ID and password are used. When the internalServerId feature is used, it does not work because runAs with system identity is not supported. You must specify RunAs roles. When internalServerID is used, use the RunAsSpecified with a user ID and password that is mapped to the administrator role. See “Administrative roles and naming service authorization” on page 566 for more information about internalServerId.

The EJB specification only supports delegation (RunAs) at the Enterprise JavaBeans (EJB) level. But an extension allows EJB method-level RunAs specification. With an EJB method level, the RunAs specification can specify a different RunAs role for different methods within the same enterprise beans.

The RunAs specification is detailed in the deployment descriptor, which is the `ejb-jar.xml` file in the EJB module and the `web.xml` file in the web module. The extension to the RunAs specification is included in the `ibm-ejb-jar-ext.xml` file.

An IBM-specific binding file is available for each application that contains a mapping from the RunAs role to the user. This file is specified in the `ibm-application-bnd`.xml file.

These specifications are read by the runtime during application startup. The following figure illustrates the delegation mechanism, as implemented in the WebSphere Application Server security model.



Delegation Process

Two tables help in the delegation process:
- Resource to RunAs role mapping table
- RunAs role to user ID and password mapping table

Use the Resource to RunAs role mapping table to get the role that is used by a servlet or by enterprise beans to propagate to the next enterprise beans call.

Use the RunAsRole to user ID and password mapping table to get the user ID that belongs to the RunAs role and its password.

Delegation is performed after successful authentication and authorization. During this process, the delegation module consults the Resource to RunAs role mapping table to get the RunAs role (3). The delegation module consults the RunAs role to user ID and password mapping table to get the user that belongs to the RunAs role (4). The user ID and password is used to create a new credential using the authentication module, which is not shown in the figure.

The resulting credential is stored in the Object Request Broker (ORB) Current as an invocation credential (5). Servlet and enterprise beans when invoking other enterprise beans pick up the invocation credential from the ORB Current (6) and call the next enterprise beans (7).

Authorizing access to Java EE resources using Tivoli Access Manager

The Java Authorization Contract for Containers (JACC) defines a contract between Java Platform, Enterprise Edition (Java EE) containers and authorization providers. You can use the default authorization or an external JACC authorization provider. When security is enabled in WebSphere Application Server, the default authorization is used unless a JACC provider is specified.

Before you begin

JACC enables any third-party authorization providers to plug into a Java EE application server (such as WebSphere Application Server) to make the authorization decisions when a Java EE resource is accessed. By default, WebSphere Application Server implements the JACC provider by using Tivoli Access Manager as the external authorization provider.

Read the following articles for more detailed information about JACC before you attempt to configure WebSphere Application Server to use a JACC provider:

Procedure

- "JACC support in WebSphere Application Server" on page 579
- "JACC providers" on page 581
- "Tivoli Access Manager integration as the JACC provider" on page 585

Using the built-in authorization provider

You can extend the capabilities of WebSphere Application Server by plugging in your own authorization provider. You can use the built-in authorization or an external JACC authorization provider.

About this task

For an explanation of the administrative console panels that support these capabilities, see:

Procedure

- Use the built-in authorization provider. It is recommended that you do not modify any settings on the authorization provider panels if you use the **Built-in authorization** option. For more information, see "External authorization provider settings."
- Use an external authorization provider. If you use the **External authorization using a JACC provider** option, the external providers must be based on the Java Authorization Contract for Containers (JACC) specification to handle the Java Platform, Enterprise Edition (Java EE) authorization. By default, WebSphere Application Server enables you to configure the Tivoli Access Manager Java Authorization Contract for Containers (JACC) provider as the default external JACC provider. For more information, see "External Java Authorization Contract for Containers provider settings" on page 599 and "Tivoli Access Manager JACC provider settings" on page 606.

External authorization provider settings

Use this page to enable a Java Authorization Contract for Containers (JACC) provider for authorization decisions.

To view this administrative console page, complete the following steps:

1. Click **Security > Global security**.
2. Click **External authorization providers**.

The application server provides a default authorization engine that performs all of the authorization decisions. In addition, the application server also supports an external authorization provider using the JACC specification to replace the default authorization engine for Java Platform, Enterprise Edition (Java EE) applications.

JACC is part of the Java EE specification, which enables third-party security providers such as Tivoli Access Manager to plug into the application server and make authorization decisions.

Important: Unless you have an external JACC provider or want to use a JACC provider for Tivoli Access Manager that can handle Java EE authorizations based on JACC, and it is configured and set up to use with the application server, do not enable **External authorization using a JACC provider**.

Built-in authorization:

Use this option all the time unless you want an external security provider such as the Tivoli Access Manager to perform the authorization decision for Java EE applications that are based on the JACC specification.

External JACC provider: Use this link to configure the application server to use an external JACC provider. For example, to configure an external JACC provider, the policy class name and the policy configuration factory class name are required by the JACC specification.

The default settings that are contained in this link are used by Tivoli Access Manager for authorization decisions. If you intend to use another provider, modify the settings as appropriate.

External Java Authorization Contract for Containers provider settings

Use this page to configure the application server to use an external Java Authorization Contract for Containers (JACC) provider. For example, the policy class name and the policy configuration factory class name are required by the JACC specification.

Use these settings when you have set up an external security provider that supports the JACC specification to work with the application server. The configuration process involves installing and configuring the provider server and configuring the client of the provider in the application server to communicate with the server. If the JACC provider is not enabled, these settings will be ignored.

To view this administrative console page, complete the following steps:

1. Click **Security** > **Global security**.
2. Click **External authorization providers**.
3. Under Authorization provider, click **External JACC provider**.

Use the default settings when you use Tivoli Access Manager as the JACC provider. Install and configure the Tivoli Access Manager server prior to using it with the application server. Use the Tivoli Access Manager properties link under Additional properties, and configure the Tivoli Access Manager client in the application server to use the Tivoli Access Manager server. If you intend to use another provider, modify the settings as appropriate.

Name:

Specifies the name that is used to identify the external JACC provider.

This field is required.

Information	**Value**
Data type:	String

Description:

Provides an optional description for the provider.

Information	**Value**
Data type:	String

Policy class name:

Specifies a fully qualified class name that represents the javax.security.jacc.policy.provider property as per the JACC specification. The class represents the provider-specific implementation of the java.security.Policy abstract methods.

The class file for the custom JACC provider must reside in the `WAS-INSTALL/lib/ext` directory. This enables the application server, node agents, and the deployment manager to operate correctly.

Do not add the Java archive (JAR) file, which contains the class file, to the `<WAS_HOME>/lib` directory in a product environment as service releases overwrite files in this directory.

This class is used during authorization decisions. The default class name is for Tivoli Access Manager implementation of the policy file.

This field is required. For information on enabling the JACC provider using this field, see the “Enabling the JACC provider for Tivoli Access Manager” article in the information center.

Information	**Value**
Data type:	String
Default:	com.tivoli.pd.as.jacc.TAMPolicy

Policy configuration factory class name:

Specifies a fully qualified class name that represents the javax.security.jacc.PolicyConfigurationFactory.provider property as per the JACC specification. The class represents the provider-specific implementation of the javax.security.jacc.PolicyConfigurationFactory abstract methods.

The class file must reside in the class path of each application server process. These processes include the application server, node agents and the deployment manager.

Do not add the Java archive (JAR) file, which contains the class file, to the `<WAS_HOME>/lib` directory in a product environment as service releases overwrite files in this directory.

This class represents the provider-specific implementation of the PolicyConfigurationFactory abstract class. This class is used to propagate the security policy information to the JACC provider during the installation of the J2EE application. The default class name is for the Tivoli Access Manager implementation of the policy configuration factory class name.

This field is required.

Information	**Value**
Data type:	String
Default:	com.tivoli.pd.as.jacc.TAMPolicyConfigurationFactory

Role configuration factory class name:

Specifies a fully qualified class name that implements the com.ibm.wsspi.security.authorization.RoleConfigurationFactory interface.

The class file must reside in the class path of each application server process. These processes include the application server, node agents and the deployment manager.

Do not add the Java archive (JAR) file, which contains the class file, to the <WAS_HOME>/lib directory in a product environment as service releases overwrite files in this directory.

When you implement this class, the authorization table information in the binding file is propagated to the provider during the installation of the J2EE application. The default class name is for the Tivoli Access Manager implementation of the role configuration factory class name.

This field is optional. For information on enabling the JACC provider using this field, see the “Enabling the JACC provider for Tivoli Access Manager” article in the information center.

Information	Value
Data type:	String
Default:	com.tivoli.pd.as.jacc.TAMRoleConfigurationFactory

Provider initialization class name:

Specifies a fully qualified class name that implements the com.ibm.wsspi.security.authorization.InitializeJACCProvider interface.

The class file must reside in the class path of each application server process. These processes include the application server, node agents and the deployment manager.

Do not add the Java archive (JAR) file, which contains the class file, to the <WAS_HOME>/lib directory in a product environment as service releases overwrite files in this directory.

When implemented, this class is called at the start and the stop of all the application server processes. You can use this class for any required initialization that is needed by the provider client code to communicate with the provider server. The properties that are entered in the custom properties link are passed to the provider when the process starts up. The default class name is for the Tivoli Access Manager implementation of the provider initialization class name.

This field is optional. For information on enabling the JACC provider using this field, see the “Enabling the JACC provider for Tivoli Access Manager” article in the information center.

Information	Value
Data type:	String
Default:	com.tivoli.pd.as.jacc.cfg.TAMConfigInitialize

Requires the EJB arguments policy context handler for access decisions:

Specifies whether the JACC provider requires the EJBArgumentsPolicyContextHandler handler to make access decisions.

Because this option has an impact on performance, do not set it unless it is required by the provider. Normally, this handler is required only when the provider supports instance-based authorization. Tivoli Access Manager does not support this option for J2EE applications.

Information	Value
Default:	Disabled

Supports dynamic module updates:

Specifies whether you can apply changes made to security policies of web modules in a running application, dynamically without affecting the rest of the application.

If this option is enabled, the security policies of the added or modified web modules are propagated to the JACC provider and only the affected web modules are started.

If this option is disabled, then the security policies of the entire application are propagated to the JACC provider for any module-level changes. The entire application is restarted for the changes to take effect.

Typically, this option is enabled for an external JACC provider.

Information	Value
Default:	Enabled

Custom properties:

Specifies the properties that are required by the provider.

These properties are propagated to the provider during the startup process when the provider initialization class name is initialized. If the provider does not implement the provider initialization class name as described previously, the properties are not used.

The Tivoli Access Manager implementation does not require that you enter any properties in this link.

Tivoli Access Manager properties:

Specifies properties that are required by the Tivoli Access Manager implementation.

These properties are used to set up the communication between the application server and the Tivoli Access Manager server. You must install and configure the Tivoli Access Manager server before entering these properties.

Enabling an external JACC provider

Use this topic to enable an external JACC provider using the administrative console.

Before you begin

The Java Authorization Contract for Containers (JACC) defines a contract between Java Platform, Enterprise Edition (Java EE) containers and authorization providers. This contract enables any third-party authorization providers to plug into a Java EE 5 application server, such as WebSphere Application Server to make the authorization decisions when a Java EE resource is accessed.

Procedure

1. From the WebSphere Application Server administrative console, click **Security > Global security > External authorization providers**.
2. Under Related items, click **External JACC provider**.
3. The fields are set for Tivoli Access Manager by default. If you do not plan to use Tivoli Access Manager as the JACC provider, replace these fields with the details for your own external JACC provider.
4. If any custom properties are required by the JACC provider, click **Custom properties** under Additional properties and enter the properties. When using the Tivoli Access Manager, use the **Tivoli Access Manager properties** link instead of the Custom properties link. For more information, see “Configuring the JACC provider for Tivoli Access Manager using the administrative console” on page 603.
5. On the External authorization providers panel, select the **External authorization using a JACC provider** option and click **OK**.

6. Complete the remaining steps to enable security. If you are using Tivoli Access Manager, you must select LDAP as the user registry and use the same LDAP server. For more information on configuring LDAP registries, see “Configuring Lightweight Directory Access Protocol user registries” on page 170.
7. Restart all servers to make these changes effective.

Configuring the JACC provider for Tivoli Access Manager using the administrative console

Use this task to configure Tivoli Access Manager as the Java Authorization Contract for Containers (JACC) provider using the administrative console.

Before you begin

Prior to completing the following steps, verify that you have previously created a security administrative user. For more information, see “Creating the security administrative user for Tivoli Access Manager” on page 605.

About this task

The following configuration is performed on the management server. When you click either **Apply** or **OK**, configuration information is checked for consistency, saved, and applied if successful.

To configure Tivoli Access Manager as the JACC provider using the administrative console, complete the following steps:

Procedure

1. Start the WebSphere Application Server administrative console by clicking `http://`*`yourhost`*`.domain:`*`port_number`*`/ibm/console` after starting WebSphere Application Server. If security is currently disabled, log in with any user ID. If security is currently enabled, log in with a predefined administrative ID and password. This ID is typically the server user ID that is specified when you configure the user registry.
2. Click **Security > Global security > External authorization providers**.
3. Under General properties, select **External authorization using a JACC provider**.
4. Under Related items, click **External JACC provider**.
5. Under Additional properties, click **Tivoli Access Manager Properties**. The Tivoli Access Manager JACC provider configuration screen is displayed.
6. Enter the following information:

   Enable embedded Tivoli Access Manager
   Select this option to enable Tivoli Access Manager.

   Ignore errors during embedded Tivoli Access Manager disablement
   Select this option when you want to unconfigure the JACC provider. Do not select this option during configuration.

   Client listening port set
   WebSphere Application Server must listen using a TCP/IP port for authorization database updates from the policy server. More than one process can run on a particular node or machine. More than one authorization server can be specified by separating the entries with commas. Specifying more than one authorization server at a time is useful for reasons of failover and performance. Enter the listening ports used by Tivoli Access Manager clients, separated by a comma. If a range of ports is specified, separate the lower and higher values by a colon (**:**) (for example, `7999, 9990:999`).

Policy server
Enter the name of the Tivoli Access Manager policy server and the connection port. Use the `policy_server:port` form. The policy communication port is set at the time of the Tivoli Access Manager configuration, and the default is 7135.

Authorization servers
Enter the name of the Tivoli Access Manager authorization server. Use the `auth_server:port:priority` form. The authorization server communication port is set at the time of the Tivoli Access Manager configuration, and the default is 7136. The priority value is determined by the order of the authorization server use (for example, `auth_server1:7136:1` and `auth_server2:7137:2`). A priority value of `1` is required when configuring against a single authorization server.

Administrator user name
Enter the Tivoli Access Manager administrator user name that was created when Tivoli Access Manager was configured; it is usually `sec_master`.

Administrator user password
Enter the Tivoli Access Manager administrator password.

User registry distinguished name suffix
Enter the distinguished name suffix for the user registry that is shared between Tivoli Access Manager and WebSphere Application Server, for example, `o=ibm, c=us`.

Security domain
You can create more than one security domain in Tivoli Access Manager, each with its own administrative user. Users, groups and other objects are created within a specific domain, and are not permitted to access resource in another domain. Enter the name of the Tivoli Access Manager security domain that is used to store WebSphere Application Server users and groups.

If a security domain is not established at the time of the Tivoli Access Manager configuration, leave the value as `Default`.

Administrator user distinguished name
Enter the full distinguished name of the WebSphere Application Server security administrator ID (for example, `cn=wasdmin, o=organization, c=country`). The ID name must match the Server user ID on the Lightweight Directory Access Protocol (LDAP) User Registry panel in the administrative console. To access the LDAP User Registry panel, click **Security > Global security**. Under **User account repository**, choose **Standalone LDAP registry** as the available realm definition. Then click **Configure**.

7. When all information is entered, click **OK** to save the configuration properties. The configuration parameters are checked for validity and the configuration is attempted at the host server or cell manager.

Results

After you click **OK**, WebSphere Application Server completes the following actions:

- Validates the configuration parameters.
- Configures the host server or cell manager.

These processes might take some time depending on network traffic or the speed of your machine.

What to do next

If the configuration is successful, the parameters are copied to all subordinate servers, including the node agents. To complete the embedded Tivoli Access Manager client configuration, you must restart all of the servers, including the host server, and enable WebSphere Application Server security.

Creating the security administrative user for Tivoli Access Manager:

Enabling security requires the creation of a WebSphere Application Server administrative user. Use the Tivoli Access Manager command-line pdadmin utility to create the Tivoli Access Manager administrative user for WebSphere Application Server. This utility is available on the policy server host machine.

About this task

Follow these steps to use the pdadmin utility.

Procedure

1. From a command line, start the pdadmin utility as the Tivoli Access Manager administrative user, sec_master:

```
pdadmin -a sec_master -p sec_master_password
```

2. Create a WebSphere Application Server security user. For example, the following instructions create a new user, *wasadmin*. The command is entered as one continuous line:

```
pdadmin> user create wasadmin cn=wasadmin,o=organization,
c=country wasadmin wasadmin myPassword
```

   Substitute values for organization and country that are valid for your Lightweight Directory Access Protocol (LDAP) user registry.

3. Enable the account for the WebSphere Application Server security administrative user by issuing the following command:

```
pdadmin> user modify wasadmin account-valid yes
```

What to do next

Configure the Java Authorization Contract for Container (JACC) provider for Tivoli Access Manager. For more information, see “Tivoli Access Manager JACC provider configuration.”

Tivoli Access Manager JACC provider configuration:

You can configure the Java Authorization Contract for Containers (JACC) provider for Tivoli Access Manager to deliver authentication and authorization protection for your applications or for authentication only. Most deployments that use the JACC provider for Tivoli Access Manager to configure Tivoli Access Manager provide both authentication and authorization functionality.

If you want Tivoli Access Manager to provide authentication, but leave authorization as part of WebSphere Application Server's native security, add the `com.tivoli.pd.as.amwas.DisableAddAuthorizationTableEntry=true` property to the `amwas.amjacc.template.properties` file. The file is located in the *profile_root*`/config/cells/`*`cell_name`* directory.

After this property is set, perform the tasks for setting Tivoli Access Manager Security, as documented.

You can configure the JACC provider for Tivoli Access Manager using either the WebSphere Application Server administrative console or the **wsadmin** command-line utility.

- For details on configuring the JACC provider for Tivoli Access Manager using the administrative console, refer to “Configuring the JACC provider for Tivoli Access Manager using the administrative console” on page 603.
- For details on configuring the Tivoli Access Manager JACC provider using the **wsadmin** command line utility, refer to Configuring the JACC provider for Tivoli Access Manager using the wsadmin utility.

The JACC configuration files for Tivoli Access Manager that are common across multiple WebSphere Application Server profiles are created by default under the `java/jre` directory. When you install WebSphere Application Server, you are given permissions to read and write to the files in this directory.

AIX HP-UX Solaris Profiles created by users who are different to the user that installed the application have read-only permissions for this directory.

This situation is not ideal because configuration of the JACC provider for Tivoli Access Manager fails in these situations. To avoid this situation, you can add the following property to the *profile_root*/config/cells/*cell_name*/amwas.amjacc.template.properties file:
com.tivoli.pd.as.jacc.CommonFileLocation=*new location* where *new location* is a fully qualified directory name.

This property applies read and write permissions to the java/jre directory.

The **wsadmin** command is available to reconfigure the Java Authorization Contract for Containers (JACC) Tivoli Access Manager interface:

```
$AdminTask reconfigureTAM -interactive
```

This command effectively prompts you through the process of unconfiguring the interface and then reconfiguring it.

Tivoli Access Manager JACC provider settings:

Use this page to configure the Java Authorization Contract for Container (JACC) provider for Tivoli Access Manager.

Note: When a third-party authorization such as Tivoli Access Manager or SAF for z/OS is used, the information in the administrative console panel might not represent the data in the provider. Also, any changes to the panel might not be reflected in the provider automatically. Follow the provider's instructions to propagate any changes made to the provider.

To view the JACC provider settings for Tivoli Access Manager, complete the following steps:

1. Click **Security > Global security**.
2. Under Authentication, click **External authorization providers**.
3. Under Authorization provider, click **External JACC provider**.
4. Click **Configure** to configure the properties for Tivoli Access Manager.

Enable embedded Tivoli Access Manager:

Enables or disables the embedded Tivoli Access Manager client configuration.

Information	**Value**
Default:	Disabled
Range:	Enabled or Disabled

Note: If you want to disable Tivoli Access Manager as the JACC provider, clear this option and also select **Default authorization**.

Ignore errors during embedded Tivoli Access Manager disablement:

Specifies whether to ignore error messages during the unconfiguration process.

If you check this check box and click **OK** or **Apply**, when you unconfigure the embedded Tivoli Access Manager, any unconfiguration errors are ignored and the process completes. If you do not check this check box, unconfiguration errors cause the unconfiguration process to stop.

This option is applicable only when re-configuring an embedded Tivoli Access Manager client or disabling an embedded Tivoli Access Manager.

Information	Value
Default:	Disabled
Range:	Enabled or Disabled

Client listening port set:

Enter the ports that are used as listening ports by Tivoli Access Manager clients.

The application server needs to listen on a TCP/IP port for authorization database updates from the policy server. More than one process can run on a particular node and machine, so a list of ports is required for use by the processes. If you specify a range of ports, separate the lower and higher values by a colon (:). The first 20% of the range is reserved for the deployment manager. Single ports and port ranges are specified on separate lines. An example list might look like the following example:

```
7999
 8900:8999
```

Policy server:

Enter the name, fully-qualified domain name, or IP address of the Tivoli Access Manager policy server and the connection port.

Use the form *policy_server*:*port*. The policy server communication port was set at the time of the Tivoli Access Manager configuration. The default is 7135.

Authorization servers:

Enter the name, fully-qualified domain name, or IP address of the Tivoli Access Manager authorization server. Use the form, *auth_server*:*port*:*priority*.

The authorization server communication port is set at the time of Tivoli Access Manager configuration. The default is 7136. You can specify more than one authorization server by entering each server on a new line. Configuring more than one authorization server provides for failover. The priority value is the order of authorization server use. For example:

```
auth_server1.mycompany.com:7136:1
auth_server2.mycompany.com:7137:2
```

A priority of 1 is still required when configuring a single authorization server.

Administrator user name:

Enter the Tivoli Access Manager administration user ID, as created at the time of Tivoli Access Manager configuration. This ID is usually, `sec_master`.

Administrator user password:

Enter the Tivoli Access Manager administration password for the user ID that is entered in the **Administrator user name** field.

User registry distinguished name suffix:

Enter the distinguished name suffix for the user registry to share between Tivoli Access Manager and the application server. For example: o=*organization*,c=*country*

Security domain:

Enter the name of the Tivoli Access Manager security domain that is used to store application server users and groups.

Specification of the Tivoli Access Manager domain is required because more than one security domain can be created in Tivoli Access Manager with its own administrative user. Users, groups, and other objects are created within a specific domain and are not permitted to access resources in another domain. If a security domain is not established at the time of Tivoli Access Manager configuration, leave the value as *Default*.

Information	**Value**
Default:	Default

Administrator user distinguished name:

Enter the fully distinguished name of the security administrator ID for the application server. For example, cn=wasadmin,o=*organization*,c=*country*

JACC provider configuration properties for Tivoli Access Manager:

The JACC provider configuration properties detailed in the following section may require configuration.

The Java property files are created in the *profile_root*/`etc/tam` directory.

Two properties files might require configuration:

- `amwas.`*`node_name_server_name`*`.amjacc.properties` contains properties that are used by the JACC provider of Tivoli Access Manager.
- `amwas.`*`node_name_server_name`*`.pdjlog.properties` contains logging properties that are created from the `amwas.pdjlog.template.properties` file for the specific node and server combination at the time of configuration.

Use `amwas.`*`node_name_server_name`*`.amjacc.properties` file to configure static role caching, dynamic role caching, object caching, and role-based policy framework properties.

Static role caching properties:

The static role cache holds role memberships that do not expire.

These properties are in the *profile_root*/`etc/tam/amwas.`*`node_name_server_name`*`.amjacc.properties` file.

The *profile_root* directory is the value of the profilePath parameter at profile creation time.

com.tivoli.pd.as.cache.EnableStaticRoleCaching=true:

Enables or disables static role caching. Static role caching is enabled by default.

com.tivoli.pd.as.cache.StaticRoleCache=com.tivoli.pd.as.cache.StaticRoleCacheImpl:

This property holds the implementation class of the static role cache. You do not need to change this property, although the opportunity exists to implement your own cache, if necessary.

com.tivoli.pd.as.cache.StaticRoleCache.Roles=Administrator,Operator,Monitor,Deployer:

Defines the administration roles for WebSphere Application Server.

Tip: Enhance Application performance by adding the static roles: **CosNamingRead**, **CosNamingWrite**, **CosNamingCreate**, **CosNamingDelete**. These roles support for improved lookup performance within the application naming service.

Dynamic role caching properties:

The dynamic role cache holds role memberships that expire.

These properties are in the *profile_root*/etc/tam/amwas.*node_name_server_name*.amjacc.properties file.

The *profile_root* directory is the value of the profilePath parameter at profile creation time.

com.tivoli.pd.as.cache.EnableDynamicRoleCaching=true:

Enables or disables dynamic role caching. Dynamic role caching is enabled by default.

com.tivoli.pd.as.cache.DynamicRoleCache=com.tivoli.pd.as.cache.DynamicRoleCacheImpl:

This property holds the implementation class of the dynamic role cache. You do not need to change this property, although the opportunity exists to implement your own cache, if necessary.

com.tivoli.pd.as.cache.DynamicRoleCache.MaxUsers=100000:

The maximum number of users that the cache supports before a cache cleanup is performed. The default number of users is 100000.

com.tivoli.pd.as.cache.DynamicRoleCache.NumBuckets=20:

The number of tables that is used internally by the dynamic role cache. The default is 20. When a large number of threads use the cache, increase the value to tune and optimize cache performance.

com.tivoli.pd.as.cache.DynamicRoleCache.PrincipalLifeTime=10:

The period of time in minutes that a principal entry is stored in the cache. The default time is 10 minutes. The term, *principal*, here refers to the Tivoli Access Manager credential that is returned from a unique Lightweight Directory Access Protocol user.

com.tivoli.pd.as.cache.DynamicRoleCache.RoleLifetime=20:

The period of time in seconds that a role is stored in the role list for a user before it is discarded. The default is 20 seconds.

Object caching properties:

The object cache is used to cache all Tivoli Access Manager objects, including their extended attributes. This bypasses the need to query the Tivoli Access Manager authorization server for each resource request.

These properties are in the *profile_root*/etc/tam/amwas.*node_name_server_name*.amjacc.properties file.

The *profile_root* directory is the value of the profilePath parameter when the profile is created.

These object cache properties cannot be changed after configuration. If any require changing, it should be done before configuration of the nodes in the cell. Changes need to be made in the template properties file before any configuration actions are performed. Properties changed after configuration might cause access decisions to fail.

com.tivoli.pd.as.cache.EnableObjectCaching=true:

This property enables or disables object caching. The default value is true.

com.tivoli.pd.as.cache.ObjectCache=com.tivoli.pd.as.cache.ObjectCacheImpl:

This property is the class used to perform object caching. You can implement your own object cache if required. This can be done by implementing the *com.tivoli.pd.as.cache.IObjectCache* interface. The default is *com.tivoli.pd.as.cache.ObjectCacheImpl*.

com.tivoli.pd.as.cache.ObjectCache.NumBuckets=20:

This property specifies the number of buckets used to store object cache entries in the underlying hash table. The default is 20.

com.tivoli.pd.as.cache.ObjectCache.MaxResources=10000:

This property specifies the total number of entries for all buckets in the cache. This figure, divided by NumBuckets determines the maximum size of each bucket. The default is 10000.

com.tivoli.pd.as.cache.ObjectCache.ResourceLifeTime=20:

This property specifies the length of time in minutes that objects are kept in the object cache. The default is 20.

Role-based policy framework properties:

Although it is very unlikely that you will need to change these properties, use this file to reference supported properties within the role-based policy framework.

The role-based policy framework parameters are located in the Java Authorization Contract for Containers (JACC) configuration file and in the authorization configuration file. They are set at the time of JACC provider configuration and authorization server configuration. The role-based policy framework settings for the authorization table and the JACC provider can be modified separately for each WebSphere Application Server instance. The `amwas.`*`node_server`*`.authztable.properties` configuration file is generated from the authorization table. The `amwas.`*`node_name_server_name`*`.amjacc.properties` configuration file is generated from the JACC provider. Both files are stored in the *profile_root*`/etc/tam` directory. It is very unlikely that you might need to change these properties. The properties are described here for reference.

The settings cannot be changed after configuration. Make changes in the template properties file before any configuration actions are performed. Properties that are changed after configuration will cause access decisions to fail.

com.tivoli.pd.as.rbpf.AMAction=i:

This property is used to signify that a user is granted access to a role. This value is added to a Tivoli Access Manager access control list (ACL) and places invoke access on roles for users and groups.

com.tivoli.pd.as.rbpf.AMActionGroup=WebAppServer:

This property sets the Tivoli Access Manager action group that serves as a container for the action that is specified by the com.tivoli.pd.as.rbpf.AMAction property. The permission set in the `com.tivoli.pd.as.rbpf.AMAction` property goes into this action group.

com.tivoli.pd.as.rbpf.PosRoot=WebAppServer:

This property is used to determine where roles are stored in the protected object space.

com.tivoli.pd.as.rbpf.ProductId=deployedResources:

This property specifies the location under the root location that is specified in the posroot property to separate other products in the protected object space. Embedded Tivoli Access Manager objects are found in the `/WebAppServer/deployedResources` directory. The default value is `deployedResources`.

com.tivoli.pd.as.rbpf.ResourceContainerName=Resources:

This property specifies the Tivoli Access Manager object space container name for the protected resources. The default location is the `/WebAppServer/deployedResources/Resources` directory.

com.tivoli.pd.as.rbpf.RoleContainerName=Roles:

This property specifies the Tivoli Access Manager protected object space container name for the security roles. The default location is the `/WebAppServer/deployedResources/Roles` directory.

System-dependent configuration properties:

Do not change these system-dependent configuration properties. These properties are included in this article for reference only.

These properties are in the *app_server_root*`/etcamwas.`*node_name_server_name*`.amjacc.properties` file.

The *profile_root* variable is the value of the profilePath parameter when the profile is created.

com.tivoli.pd.as.rbpf.AmasSession.CfgURL=file/:$WAS_HOME/profiles/profile_name/etc/tam/amwas.node_server.pdperm.properties:

This entry is generated by the Java Authorization Contract for Containers (JACC) provider configuration. This argument specifies the location of the file that contains information about the JACC provider of Tivoli Access Manager. Do not change this entry or the properties in the amwas.node_server.pdperm.properties file.

com.tivoli.pd.as.rbpf.AmasSession.CfgURL=file/:user_root/etc/tam/amwas.node_server.pdperm.properties:

This entry is generated by the Java Authorization Contract for Containers (JACC) provider configuration. It specifies the location of the file that contains information about the Tivoli Access Manager JACC provider. Do not change this entry or the properties in the amwas.node_server.pdperm.properties file.

com.tivoli.pd.as.rbpf.AmasSession.LoggingURL=file/:$WAS_HOME/profiles/profile_name/etc/tam/amwas.node_server.pdjlog.properties:

This entry contains the location of the logging configuration file for the JACC provider of Tivoli Access Manager. The referenced file is generated by the JACC provider of Tivoli Access Manager configuration. Do not change this entry.

com.tivoli.pd.as.rbpf.AmasSession.LoggingURL=file/:user_root/etc/tam/amwas.node_server.pdjlog.properties:

This entry contains the location of the logging configuration file for the Tivoli Access Manager JACC provider. The file referenced is generated by the Tivoli Access Manager JACC provider configuration. Do not change this entry.

Administering security users and roles with Tivoli Access Manager

Use these steps to manage user-to-role mappings and user-to-group mappings for applications.

About this task

User-to-role mapping and user-to-group mapping for the JACC provider of Tivoli Access Manager are performed using the WebSphere Application Server administrative console.

Procedure

1. Click **Applications > Enterprise applications >** *application_name*.
2. Under Additional properties, click **Security role to user/group mapping**. The user and groups management screen is displayed.
3. Select the role that requires user or group management and use **Lookup users** or **Lookup groups** to manage the users or groups for the selected role. The native role mapping uses the MapRolesToUsers administrative task. If you are using Tivoli Access Manager, use the TAMMapRolesToUsers administrative task instead. The syntax and options for the Tivoli version are the same as those used in the native version. For more information, see Role-based security with embedded Tivoli Access Manager and Configuring Tivoli Access Manager groups.

Configuring Tivoli Access Manager groups

Use these steps to configure the WebSphere Application Server administrative console to add objects of the accessGroup class to the list of object classes that represent user registry groups.

About this task

You can use the WebSphere Application Server administrative console to specify security policies for applications that run in the WebSphere Application Server environment. You can also use the WebSphere Application Server administrative console to specify security policies for other web resources, based on the entities that are stored in the user registry.

Tivoli Access Manager adds the accessGroup object class to the registry. Tivoli Access Manager administrators can use the pdadmin utility, which is available only on the policy server host in the PD.RTE fileset, to create new groups. These new groups are added to the registry as the accessGroup object class.

The WebSphere Application Server administrative console is not configured by default to recognize objects of the accessGroup class as user registry groups. You can configure the WebSphere Application Server administrative console to add this object class to the list of object classes that represent user registry groups. To do this configuration, complete the following instructions:

Procedure

1. From the WebSphere Application Server administrative console, access the advanced settings for configuring security by clicking **Security > Global security**.
2. Under User account repository, click the **Available realm definitions** drop-down list, select **Standalone LDAP registry**, and click **Configure**.
3. Under Additional properties, click **Advanced Lightweight Directory Access Protocol (LDAP) user registry settings**.
4. Modify the **Group Filter** field. Add the following entry: `(objectclass=accessGroup)`

   The Group Filter field looks like the following example:

   ```
   (&(cn=%w)(|(objectclass=groupOfNames)
   (objectclass=groupOfUniqueNames)(objectclass=accessGroup)))
   ```
5. Modify the **Group Member ID Map** field. Add the following entry: `accessGroup:member`. The Group Member ID Map field looks like the following example:

   ```
   groupOfNames:member;groupOfUniqueNames:uniqueMember;
   accessGroup:member
   ```

6. Stop and restart WebSphere Application Server.

Configuring additional authorization servers for Tivoli Access Manager

Tivoli Access Manager secure domains can contain more than one authorization server. Having multiple authorization servers is useful for providing a failover capability as well as improving performance when the volume of access requests is large.

Procedure

1. Refer to the *Tivoli Access Manager Base Administration Guide* for details on installing and configuring authorization servers. This document is available in the IBM Tivoli Access Manager for e-business information center.
2. Re-configure the Java Authorization Contract for Containers (JACC) provider using the `$AdminTask reconfigureTAM interactive` **wsadmin** command. Enter all new and existing options. The following table lists the information that you are asked to provide for the reconfigureTAM command. The table also lists the properties that apply to the configureTAM and unconfigureTAM commands.

Table 79. Commands for configuring, reconfiguring, and unconfiguring Tivoli Access Manager. The following table lists the information that you are asked to provide for the configureTAM command. The table also lists the properties that apply to the unconfigureTAM and reconfigureTAM commands.

Property	Default	Relevant command	Description
Websphere Application Server node name	*	• configureTAM • reconfigureTAM • unconfigureTAM	Specify a single node on which to run the configuration task.
Tivoli Access Manager Policy Server	Default port: 7135	• configureTAM • reconfigureTAM	Enter the name of the Tivoli Access Manager policy server and the connection port. Use the format, *policy_server* **:** *port*. The policy server communication port is set at the time of Tivoli Access Manager configuration.
Tivoli Access Manager Authorization Server	Default port: 7136	• configureTAM • reconfigureTAM	Enter the name, port, and priority of each configured Tivoli Access Manager authorization server. Use the format *auth_server* **:** *port* **:** *priority*. The authorization server communication port is set at the time of Tivoli Access Manager configuration. You can specify more than one authorization server by separating the entries with commas. Having more than one authorization server configured is useful for failover and performance. The priority value is the order of authorization server use. For example: auth_server**1**:7136:1,auth_server**2**:7137:2. A priority of 1 is still required when you use a single authorization server.
Websphere Application Server administrator's distinguished name		• configureTAM • reconfigureTAM	Enter the full distinguished name of the security primary administrator ID for WebSphere Application Server as created in the "Creating the security administrative user" topic in the *Securing applications and their environment* PDF. For example: cn=wasadmin,o=*organization*,c=*country*
Tivoli Access Manager user registry distinguished name suffix		• configureTAM • reconfigureTAM	Enter the suffix that you have set up in the user registry to contain the user and groups for Tivoli Access Manager. For example: o=*organization*,c=*country*
Tivoli Access Manager administrator's user name	sec_master	• configureTAM • reconfigureTAM • unconfigureTAM	Enter the Tivoli Access Manager administration user ID that you created when you configured Tivoli Access Manager. This ID is usually sec_master.
Tivoli Access Manager administrator's user password		• configureTAM • reconfigureTAM • unconfigureTAM	Enter the password that is associated with the Tivoli Access Manager administration user ID.
Tivoli Access Manager security domain	Default	• configureTAM • reconfigureTAM	Enter the name of the Tivoli Access Manager security domain that is used to store users and groups. If a security domain is not already established at the time of Tivoli Access Manager configuration, click **Return** to accept the default.
Embedded Tivoli Access Manager listening port set	8900:8999	• configureTAM • reconfigureTAM	WebSphere Application Server needs to listen on a TCP/IP port for authorization database updates from the policy server. More than one process can run on a particular node and machine so a list of ports is required for the processes. Enter the ports that are used as listening ports by Tivoli Access Manager clients, separated by a comma. If you specify a range of ports, separate the lower and higher values by a colon. For example, 7999, 9990:9999.
Defer	No	• configureTAM • reconfigureTAM • unconfigureTAM	Set this option to *yes* if you want to defer the configuration of the management server until the next restart. Set the option to *no* if you want the configuration of the management server to occur immediately. Managed servers are configured on their next restart.
Force	No	• reconfigureTAM • unconfigureTAM	Set this value to *yes* if you want to ignore errors during the unconfiguration process and allow the entire process to complete. Set the value to *no* if you want errors to stop the unconfiguration process. This option is especially useful if the environment needs to be cleaned up and problems are occurring that do not allow the entire cleanup process to complete successfully.

Logging Tivoli Access Manager security

Use this topic to enable the trace specification to indicate tracing at the required level.

About this task

The Java Authorization Contract for Containers (JACC) for Tivoli Access Manager provider messages are logged to the configured trace output location, and messages are written to standard out `SystemOut.log` file. When trace is enabled, all logging, both trace and messaging, is sent to the `trace.log` file.

Note: This topic references one or more of the application server log files. As a recommended alternative, you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log` , `SystemErr.log`, `trace.log`, and `activity.log` files on distributed and IBM i systems. You can also use HPEL in conjunction with your native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

Procedure

1. The `amwas.`*`node_server`*`.pdjlog.properties` file must be updated and the **isLogging** attribute set to *true* for the required component. For example, to enable tracing for the JACC provider for Tivoli Access Manager, set the following line to *true*:
   `amwas.`*`node_server`*`.pdjlog.properties:baseGroup.AMWASWebTraceLogger.isLogging=true`
2. Enable tracing for the JACC provider of Tivoli Access Manager components in the WebSphere Application Server administrative console by completing the following steps:
   a. Click **Troubleshooting** > **Logs and Trace** > *server_name*.
   b. Under Logs and Trace tasks, click **Diagnostic trace**.
   c. Select the **Enable Log** option.
   d. Click **Apply**.
   e. Click **Troubleshooting** > **Logs and Trace** > *server_name*.
   f. Click **Change Log Detail Levels**.
   g. Click **Components**. Tracing for all components can be enabled using the **com.tivoli.pd.as.*** command. Tracing for separate components can be enabled using the following commands:
      - **com.tivoli.pd.as.rbpf.*** for role-based policy framework tracing
      - **com.tivoli.pd.as.jacc.*** for JACC provider tracing
      - **com.tivoli.pd.as.pdwas.*** for the authorization table
      - **com.tivoli.pd.as.cfg.*** for configuration
      - **com.tivoli.pd.as.cache.*** for caching

      For more information, see Log level settings.
   h. Click **Apply**.

What to do next

The trace specification now indicates that tracing is enabled at the required level. Save the configuration and restart the server for the changes to take effect.

Tivoli Access Manager loggers:

The Java Authorization Contract for Containers (JACC) provider for Tivoli Access Manager uses the JLog logging framework as does the Java runtime environment for Tivoli Access Manager. You can enable tracing and messaging selectively for specific JACC providers for Tivoli Access Manager components.

The JACC for Tivoli Access Manager provider messages are logged to the configured trace output location, and messages are written to standard out `SystemOut.log` file. When trace is enabled, all logging, both trace and messaging, is sent to the `trace.log` file.

Note: This topic references one or more of the application server log files. As a recommended alternative, you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log` , `SystemErr.log`, `trace.log`, and `activity.log` files on distributed and IBM i systems. You can also use HPEL in conjunction with your native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

Tracing and message logging for the JACC provider for Tivoli Access Manager are configured in the `amwas.`*`node_server`*`.pdjlog.properties` properties file, which is located in the `profile_root/etc/tam` directory. This file contains logging properties from the `amwas.pdjlog.template.properties` template file for the specific node and server combination at the time of JACC provider for Tivoli Access Manager configuration.

The contents of this file let the user control:

- Whether tracing is enabled or disabled for the JACC provider of Tivoli Access Manager components.
- Whether message logging is enabled or disabled for the JACC provider of Tivoli Access Manager components.

The `amwas.`*`node_server`*`.pdjlog.properties` file defines several loggers, each of which is associated with one JACC provider of Tivoli Access Manager component. These loggers include:

Table 80. Tivoli Access Manager loggers. This table describes the Tivoli Access Manager loggers.

Logger Name	Description
AmasRBPFTraceLogger AmasRBPFMessageLogger	Logs messages and trace for the role-based policy framework. This underlying framework is used by embedded Tivoli Access Manager to make access decisions.
AmasCacheTraceLogger AmasCacheMessageLogger	Logs messages and trace for the policy caches that are used by the role-based policy framework.
AMWASWebTraceLogger AMWASWebMessageLogger	Logs messages and trace for the WebSphere Application Server authorization plug-in.
AMWASConfigTraceLogger AMWASConfigMessageLogger	Logs messages and trace for the configuration actions of the JACC provider for Tivoli Access Manager .
JACCTraceLogger JACCMessageLogger	Logs messages and trace for the JACC provider activity of Tivoli Access Manager .

Note: Tracing can have a significant impact on system performance. Enable tracing only when diagnosing the cause of a problem.

The implementation of these loggers routes messages to the WebSphere Application Server logging sub-system. All messages are written to the WebSphere Application Server `trace.log` file.

For each logger, the `amwas.`*`node_server`*`.pdjlog.properties` file defines an isLogging attribute which, when set to *true*, enables logging for the specific component. A value of *false* disables logging for that component.

The `amwas.node_server.pdjlog.properties` file defines the parent loggers MessageLogger and TraceLogger that also have an isLogging attribute. If the child loggers do not specify this isLogging attribute, they inherit the value of their respective parent. When the JACC provider for Tivoli Access Manager is enabled, the isLogging attribute is set to *true* for the MessageLogger and set to *false* for the TraceLogger logger. Message logging is enabled for all components and tracing is disabled for all components, by default.

To turn on tracing for a JACC provider component, see Logging Tivoli Access Manager security.

Interfaces that support JACC

WebSphere Application Server provides the RoleConfigurationFactory and the RoleConfiguration interfaces, which are similar to PolicyConfigurationFactory and PolicyConfiguration interfaces so the information that is stored in the bindings file can be propagated to the provider during installation. The implementation of these interfaces is optional.

RoleConfiguration interface:

Use the RoleConfiguration interface to propagate the authorization information to the provider. This interface is similar to the PolicyConfiguration interface that is found in Java Authorization Contact for Containers (JACC).

```
RoleConfiguration
      - com.ibm.wsspi.security.authorization.RoleConfiguration

/**
 * This interface is used to propagate the authorization table information
 * in the binding file during application installation. Implementation of this interface is
 * optional. When a JACC provider implements this interface during an application, both
 * the policy and the authorization table information are propagated to the provider.
* If this is not implemented, only the policy information is propagated as per
 * the JACC specification.
 * @ibm-spi
 * @ibm-support-class-A1
 */


public interface RoleConfiguration

/**
 * Add the users to the role in RoleConfiguration.
 * The role is created, if it does not exist in RoleConfiguration.
 * @param role the role name.
 * @param users the list of the user names.
 * @exception RoleConfigurationException if the users cannot be added.
 */
 public void addUsersToRole(String role, List users)
 throws RoleConfigurationException

/**
 * Remove the users to the role in RoleConfiguration.
 * @param role the role name.
 * @param users the list of the user names.
 * @exception RoleConfigurationException if the users cannot be removed.
 */
 public void removeUsersFromRole(String role, List users)
 throws RoleConfigurationException

/**
 * Add the groups to the role in RoleConfiguration.
 * The role is created if it does not exist in RoleConfiguration.
 * @param role the role name.
 * @param groups the list of the group names.
 * @exception RoleConfigurationException if the groups cannot be added.
 */
 public void addGroupsToRole(String role, List groups)
 throws RoleConfigurationException

/**
 * Remove the groups to the role in RoleConfiguration.
 * @param role the role name.
 * @param groups the list of the group names.
 * @exception RoleConfigurationException if the groups cannot be removed.
 */
 public void removeGroupsFromRole( String role, List groups)
 throws RoleConfigurationException

/**
 * Add the everyone to the role in RoleConfiguration.
 * The role is created if it does not exist in RoleConfiguration.
 * @param role the role name.
 * @exception RoleConfigurationException if the everyone cannot be added.
 */
 public void addEveryoneToRole(String role)
 throws RoleConfigurationException

/**
 * Remove the everyone to the role in RoleConfiguration.
 * @param role the role name.
 * @exception RoleConfigurationException if the everyone cannot be removed.
 */
 public void removeEveryoneFromRole( String role)
 throws RoleConfigurationException

/**
 * Add the all authenticated users to the role in RoleConfiguration.
 * The role is created if it does not exist in RoleConfiguration.
```

```
 * @param role the role name.
 * @exception RoleConfigurationException if the authentication users cannot
 *  be added.
 */
 public void addAuthenticatedUsersToRole(String role)
 throws RoleConfigurationException

/**
 * Remove the all authenticated users to the role in RoleConfiguration.
 * @param role the role name.
 * @exception RoleConfigurationException if the authentication users cannot
 * be removed.
 */
 public void removeAuthenticatedUsersFromRole( String role)
 throws RoleConfigurationException

/**
 * This commits the changes in Roleconfiguration.
 * @exception RoleConfigurationException if the changes cannot be
 * committed.
 */
 public void commit( )
 throws RoleConfigurationException

/**
 * This deletes the RoleConfiguration from the RoleConfiguration Factory.
 * @exception RoleConfigurationException if the RoleConfiguration cannot
 * be deleted.
 */
 public void delete( )
 throws RoleConfigurationException

/**
 * This returns the contextID of the RoleConfiguration.
 * @exception RoleConfigurationException if the contextID cannot be
 * obtained.
 */
 public String getContextID( )
 throws RoleConfigurationException
```

RoleConfigurationFactory interface:

The RoleConfigurationFactory interface is similar to the PolicyConfigurationFactory interface that is introduced by JACC, and is used to obtain RoleConfiguration objects based on the contextID IDs.

```
RoleConfigurationFactory
 - com.ibm.wsspi.security.authorization.RoleConfigurationFactory

/**
 * This interface is used to instantiate the com.ibm.wsspi.security.authorization.RoleConfiguration
 * objects based on the context identifier similar to the policy context identifier.
 * Implementation of this interface is required only if the RoleConfiguration interface is implemented.
 *
 * @ibm-spi
 * @ibm-support-class-A1
 */

public interface RoleConfigurationFactory
/**
 * This gets a RoleConfiguration with contextID from the
 * RoleConfigurationfactory. If the RoleConfiguration does not exist
 * for the contextID in the RoleConfigurationFactory, a new
 * RoleConfiguration with contextID is created in the
 * RoleConfigurationFactory. The contextID is similar to
 * PolicyContextID, but it does not contain the module name.
 * If remove is true, the old RoleConfiguration is removed and a new
 * RoleConfiguration is created, and returns with the contextID.
 * @return the RoleConfiguration object for this contextID
 * @param contextID the context ID of RoleConfiguration
 * @param remove true or false
 * @exception RoleConfigurationException if RoleConfiguration
 * cannot be obtained.
 **/
public abstract com.ibm.ws.security.policy.RoleConfiguration
         getRoleConfiguration(String contextID, boolean remove)
      throws RoleConfigurationException
```

InitializeJACCProvider provider:

When implemented by the provider, this interface is called by every process where the JACC provider can be used for authorization. All additional properties that are entered during the authorization check are passed to the provider. For example, the provider can use this information to initialize client code to communicate with their server or repository. The cleanup method is called during server shutdown to clean up the configuration.

Declaration:

public interface **InitializeJACCProvider**

Description:

This interface has two methods. The JACC provider can implement the interface, and WebSphere Application Server calls it to initialize the JACC provider. The name of the implementation class is obtained from the value of the initializeJACCProviderClassName system property.

This class must reside in a Java archive (JAR) file on the class path of each server that uses this provider.

```
InitializeJACCProvider
   - com.ibm.wsspi.security.authorization.InitializeJACCProvider

  /**
   * Initializes the JACC provider
      * @return 0 for success.
   * @param props the custom properties that are included for this provider will
   * pass to the implementation class.
   * @exception Exception for any problems encountered.
   **/
  public int initialize(java.util.Properties props)
  throws Exception

  /**
   * This method is for the JACC provider cleanup and will be called during a process stop.
   **/
  public void cleanup()
```

Enabling the JACC provider for Tivoli Access Manager

The Java Authorization Contract for Container (JACC) provider for Tivoli Access Manager is configured by default. Use this topic to enable the JACC provider for Tivoli Access Manager.

About this task

Restriction: Do not perform this task if you are configuring the JACC provider for Tivoli Access Manager to supply authentication services only. Only perform this task for installations that require both Tivoli Access Manager authentication and authorization protection.

The JACC provider for Tivoli Access Manager is configured by default. To enable the JACC provider for Tivoli Access Manager, complete the following steps:

Procedure

1. Click **Security > Global security > External authorization providers**.
2. Select the **External authorization using a JACC provider** option, then click **Apply**.
3. Under Related Items, click **External JACC provider**. The JACC provider settings for Tivoli Access Manager are displayed.
4. Verify that the correct settings are present to work with your Tivoli Access Manager configuration. The following list shows the JACC provider configuration settings for Tivoli Access Manager.

Table 81. JACC provider configuration settings for Tivoli Access Manager. This table describes the JACC provider configuration settings for Tivoli Access Manager.

Field	Value
Name	Tivoli Access Manager
Description	This field is optional and used as a reference.
J2EE policy class name	`com.tivoli.pd.as.jacc.TAMPolicy`
Policy configuration factory class name	`com.tivoli.pd.as.jacc.TAMPolicyConfigurationFactory`
Role configuration factory class name	`com.tivoli.pd.as.jacc.TAMRoleConfigurationFactory`
JACC provider initialization class name	`com.tivoli.pd.as.jacc.cfg.TAMConfigInitialize`
Requires the EJB arguments policy context handler for access decisions	false

Table 81. JACC provider configuration settings for Tivoli Access Manager (continued). This table describes the JACC provider configuration settings for Tivoli Access Manager.

Field	Value
Supports dynamic module updates	true

For more information, see “External Java Authorization Contract for Containers provider settings” on page 599.

5. Under Additional properties, click **Tivoli Access Manager properties** and set the properties that are associated with the embedded Tivoli Access Manager. The following table explains the properties that are needed for the embedded Tivoli Access Manager. Some fields do not have default values.

Table 82. Tivoli Access Manger properties. This table lists the Tivoli Access Manger properties.

Name	Default value	Description
Enable embedded Tivoli Access Manager	Unchecked	When you select this check box, the embedded Tivoli Access Manager is configured or reconfigured. When you clear this check box, the embedded Tivoli Access Manager is unconfigured.
Ignore errors during embedded Tivoli Access Manager disablement	Unchecked	If you check this check box and click **OK** or **Apply**, when you unconfigure the embedded Tivoli Access Manager, any unconfiguration errors are ignored and the process completes. If you do not check this check box, unconfiguration errors cause the unconfiguration process to stop.
Client listening port	8900:8999	When the embedded Tivoli Access Manager is configured and running, it requires several ports to listen for updates to the access control list database for Tivoli Access Manager. The value in this field is a range of port numbers that Tivoli Access Manager can use for this purpose. The first 20% of this range is reserved for the deployment manager. You can enter multiple ranges or individual port numbers in a line separated list. For example: 8900:8999 9100:9200 9999
Policy server		This field value specifies the name and port number of the configure and running Tivoli Access Manager policy server. The format is server:port For example:`snapper.ibm.com:7135`
Authorization servers		This field contains the names, port numbers, and priorities of all of the configured and running Tivoli Access Manager authorization servers. This field must contain at least one authorization server. If multiple authorization servers are listed, those servers are used for failover. The server with priority 1 is used first with failover to server priority 2 and so on. The format is server:port:priority with each authorization server listed on a different line. For example: snapper.ibm.com:7136:1 turtle.ibm.com:7136:2
Authorization user name	sec_master	This field value specifies the administrative user name for Tivoli Access Manager.
Administrator user password		This field value specifies the password for Tivoli Access Manager.
User registry distinguished name suffix		This field value is the suffix that is set up in the user registry to contain the users and groups for Tivoli Access Manager. For example using IBM Tivoli Directory Server: o=ibm,c=au
Security domain	Default	This field value specifies the configured security domain to use for the embedded Tivoli Access Manager.
Administrator user distinguished name		This field specifies the fully distinguished user name of the primary administrative user for WebSphere Application Server security. For example using IBM Tivoli Directory Server: cn=wasadmin,o=ibm,c=au

For more information, see “Tivoli Access Manager JACC provider settings” on page 606.

6. Click **OK**.
7. Save the settings by clicking **Save** at the top of the page.
8. Log out of the WebSphere Application Server administrative console.
9. Restart WebSphere Application Server. The security configuration is now replicated to managed servers and node agents. These other servers within a cell also require restarting before the security changes take effect.

Enabling embedded Tivoli Access Manager

Embedded Tivoli Access Manager is not enabled by default, and you need to configure it for use.

About this task

Enabling Tivoli Access Manager security within WebSphere Application Server requires:

- A supported Lightweight Directory Access Protocol (LDAP) installed somewhere on your network. This user registry contains the user and group information for both Tivoli Access Manager and WebSphere Application Server.
- Tivoli Access Manager server exists and is configured to use the user registry. For details on the installation and configuration of Tivoli Access Manager, refer to the IBM Tivoli Access Manager for e-business information center.

  Note: WebSphere Application Server contains an embedded client for Tivoli Access Manager. To use Tivoli Access Manager, you must also configure the Tivoli Access Manager server.

  However, the server version must be the same version or later as the client version. For information on the supported version of Tivoli Access Manager, see WebSphere Application Server - Supported Prerequisites.
- WebSphere Application Server is installed either in a single server model or as WebSphere Application Server, Network Deployment.
- When administrative security is configured with a Federal Information Processing Standard (FIPS) provider, the Tivoli Access Manager server must be configured for FIPS as well

Complete the following steps to enable embedded Tivoli Access Manager security:

Procedure

1. Create the security administrative user.

   For more information, see the *Securing applications and their environment* PDF.
2. Configure the Java Authorization Contract for Containers (JACC) provider for Tivoli Access Manager .

   For more information, see the *Securing applications and their environment* PDF.
3. Enable WebSphere Application Server security. When you are using Tivoli Access Manager you must configure LDAP as the user registry.

   For more information, see the *Securing applications and their environment* PDF.
4. Enable the JACC provider for Tivoli Access Manager.

   For more information, see the *Securing applications and their environment* PDF.

TAMConfig command group for the AdminTask object

You can use the Jython or Jacl scripting languages to configure embedded IBM Tivoli Access Manager with the wsadmin tool. The commands and parameters in the TAMConfig group can be used to configure or unconfigure Tivoli Access Manager.

The TAMConfig command group for the AdminTask object includes the following commands:

configureTAM

Use the configureTAM command to manually configure the Tivoli Access Manager.

Target object

None.

Required parameters

None.

Optional parameters

None.

Examples

Interactive mode example usage:

- Using Jacl:

```
$AdminTask configureTAM {-interactive}
```

- Using Jython:

```
AdminTask.configureTAM('-interactive')
```

listTAMSettings

The listSSLRepertoires command displays the current embedded Tivoli Access Manager configuration settings.

Target object

None.

Required parameters

None.

Optional parameters

None.

Examples

Interactive mode example usage:

- Using Jacl:

```
$AdminTask listTAMSettings {-interactive}
```

- Using Jython:

```
print AdminTask.listTAMSettings('-interactive')
```

modifyTAM

The modifyTAM command modifies embedded Tivoli Access Manager configuration settings.

Target object

None.

Required parameters

-adminPasswd
Specifies the Tivoli Access Manager administrator password. (String, required)

Optional parameters

-adminUid
Specifies the Tivoli Access Manager user name. (String, optional)

-nodeName
Specifies the target node or nodes. Set the value as the * asterisk character to specify all nodes. (String, optional)

Examples

Interactive mode example usage:

- Using Jacl:

```
$AdminTask modifyTAM {-adminPasswd my11password}
```

- Using Jython:

```
AdminTask.modifyTAM('-adminPasswd my11password')
```

- Using Jython list:

```
AdminTask.modifyTAM(['-adminPasswd', 'my11password'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask modifyTAM {-interactive}
```

- Using Jython:

```
AdminTask.modifyTAM('-interactive')
```

reconfigureTAM

The reconfigureTAM command reconfigures the Java Authorization Contract for Containers (JACC) Tivoli Access Manager settings.

Target object

None.

Required parameters

None.

Optional parameters

None.

Examples

Interactive mode example usage:

- Using Jacl:

```
$AdminTask reconfigureTAM {-interactive}
```

- Using Jython:

```
AdminTask.reconfigureTAM('-interactive')
```

unconfigureTAM

The unconfigureTAM command removes configuration data for the Java Authorization Contract for Containers (JACC) Tivoli Access Manager.

Required parameters

None.

Optional parameters

None.

Examples

Interactive mode example usage:

- Using Jacl:

```
$AdminTask unconfigureTAM {-interactive}
```

- Using Jython:

```
AdminTask.unconfigureTAM('-interactive')
```

configureTAMTAI

The configureTAMTAI command configures the embedded Tivoli Access Manager trust association interceptor (TAI) with classname TAMTrustAsociationInterceptorPlus.

Target object

None.

Required parameters

-policySvr
This property specifies the name of the Tivoli Access Manager policy server with which the application server communicates. The server is specified by a fully-qualified host name, the SSL port number, and the rank. The default SSL port number is 7135. For example: *myauth.mycompany.com:7135:1*.

-authSvrs
This property specifies the name of the Tivoli Access Manager authorization server with which the application server communicates. The server is specified by a fully-qualified host name, the SSL port number, and the rank. The default SSL port number is 7136. For example: *myauth.mycompany.com:7136:1*. You can specify multiple servers if the entries are separated by a comma (,).

-adminPasswd
This property specifies the password of the Tivoli Access Manager administrator user that is associated with the -adminUid parameter. The password restrictions depend upon the password policy for your Tivoli Access Manager configuration.

-loginId
The WebSEAL trusted user as created in “Creating a trusted user account in Tivoli Access Manager”.

See the Configuring single sign-on using trust association interceptor ++ article for more information. The format of the username is the short name representation.

Optional parameters

-adminUid
This property specifies the Tivoli Access Manager administrator name. If this option is not specified, `sec_master` is the default. A valid administrative ID is an alphanumeric, case-sensitive string. String values are expected to be characters that are part of the local code set. You cannot use a space in the administrative ID.

For example, for U.S. English, the valid characters are the letters a-Z, the numbers 0-9, a period (.), an underscore (_), a plus sign (+), a hyphen (-), an at sign (@), an ampersand (&), and an asterisk (*). The minimum and maximum lengths of the administrative ID, if there are limits, are imposed by the underlying registry.

-secDomain
This property specifies the Tivoli Access Manager domain name to which the administrator is authenticated. This domain must exist and an administrator ID and password must be valid for this domain. The application server is specified in this domain. If the application server is not specified, the default value is `Default`. The local domain value is retrieved from the configuration file.

A valid domain name is an alphanumeric, case-sensitive string. String values are expected to be characters that are part of the local code set. You cannot use a space in the domain name.

For example, for U.S. English, the valid characters for domain names are the letters a-Z, the numbers 0-9, a period (.), an underscore (_), a plus sign (+), a hyphen (-), an at sign (@), an ampersand (&), and an asterisk (*). The minimum and maximum lengths of the domain name, if there are limits, are imposed by the underlying registry.

-checkViaHeader
You can configure TAI so that the via header can be ignored when validating trust for a request. Set this property to `false` if none of the hosts in the via header need to be trusted. When set to false, you do not need to set the trusted host names and host ports properties. The only mandatory property to check when the via header is false is com.ibm.websphere.security.webseal.loginId. The default value of the check via header property is `false`. When using Tivoli Access Manager plug-in for web servers, set this property to false.

Note: The via header is part of the standard HTTP header that records the server names that the request passed through.

-id
This property specifies a comma-separated list of headers that exists in the request. If all of the configured headers do not exist in the request, trust cannot be established. The default value for the ID property is `iv-creds`. Any other values set in WebSphere Application Server are added to the list along with iv-creds, separated by commas.

-hostnames
Do not set this property if you are using the Tivoli Access Manager plug-in for web servers. This property specifies the host names (case-sensitive) that are both trusted and expected in the request header. Requests arriving from unlisted hosts might not be trusted. If the checkViaHeader property is not set, or is set to `false`, then the trusted host names property has no influence. If the checkViaHeader property is set to `true`, and the trusted host names property is not set, the TAI initialization fails.

-ports
Do not set this property if you are using the Tivoli Access Manager plug-in for web servers. This property is a comma-separated list of trusted host ports. Requests that arrive from unlisted ports might not be trusted. If the checkViaHeader property is not set, or is set to `false`, then this property has no

influence. If the checkViaHeader property is set to `true`, and the trusted host ports property is not set in WebSphere Application Server, the TAI initialization fails.

-viaDepth
This property indicates a positive integer that specifies the number of source hosts in the via header to check for trust. By default, every host in the via header is checked, and if any host is not trusted, trust cannot be established. The viaDepth property is used when only some of the hosts in the via header have to be trusted. The setting indicates the number of hosts that are required to be trusted.

For example, consider the following header:

If in via: HTTP/1.1 webseal1:7002, 1.1 webseal2:7001If the viaDepth property is not set, is set to 2 or is set to 0, and a request with the previous via header is received then both webseal1:7002 and webseal2:7001 need to be trusted. The following configuration then applies:

```
com.ibm.websphere.security.webseal.hostnames = webseal1,webseal2
```

If in com.ibm.websphere.security.webseal.ports = 7002,7001If the viaDepth property is set to 1, and the previous request is received, then only the last host in the via header needs to be trusted. The following configuration then applies:

```
com.ibm.websphere.security.webseal.hostnames = webseal2
com.ibm.websphere.security.webseal.ports = 7001
```

The viaDepth property is set to 0 by default, which means that all of the hosts in the via header are checked for trust.

-ssoPwdExpiry
After trust is established for a request, the single sign-on user password is cached, eliminating the need to have the TAI re-authenticate the single sign-on user with Tivoli Access Manager for every request. You can modify the cache timeout period by setting the single sign-on password expiry property to the required time in seconds. If the password expiry property is set to 0, the cached password never expires. The default value for the password expiry property is 600.

-ignoreProxy
This property can be used to tell the TAI to ignore proxies as trusted hosts. If set to `true` the comments field of the hosts entry in the via header is checked to determine if a host is a proxy. Remember that not all proxies insert comments in the via header indicating that they are proxies. The default value of the ignoreProxy property is `false`. If the checkViaHeader property is set to false, then the ignoreProxy property has no influence in establishing trust.

-configURL
For the TAI to establish trust for a request, it requires that the SvrSslCfg task be run for the Java Virtual Machine on the Application Server and result in the creation of a properties file. If this properties file is not at the default URL, which is file://java.home/PdPerm.properties, the correct URL of the properties file must be set in the configuration URL property. If this property is not set, and the SvrSslCfg-generated properties file is not in the default location, the TAI initialization fails. The default value for the config URL property is file://${WAS_INSTALL_ROOT}/java/jre/PdPerm.properties.

-defer
This property indicates whether the Tivoli Access Manager configuration portion of this task should be run immediately or deferred until the startup of the WebSphere Application Server. The default value is `no`.

Note: The TAI properties are updated immediately regardless of this setting.

Examples

Interactive mode example usage:

- Using Jacl:

```
$AdminTask configureTAMTAI {-interactive}
```

- Using Jython:

```
AdminTask.configureTAMTAI('-interactive')
```

unconfigureTAMTAI

The unconfigureTAMTAI command unconfigures the embedded Tivoli Access Manager Trust Association Interceptor with classname TAMTrustAsociationInterceptorPlus. This task does not include removing any custom properties from the security configuration.

Target object

None.

Required parameters

-adminPasswd
Specifies the password of the Tivoli Access Manager administrator user that is associated with the -adminUid parameter. The password restrictions depend upon the password policy for your Tivoli Access Manager configuration.

Optional parameters

-adminUid
Specifies the Tivoli Access Manager administrator name. If this option is not specified, `sec_master` is the default. A valid administrative ID is an alphanumeric, case-sensitive string. String values are expected to be characters that are part of the local code set. You cannot use a space in the administrative ID.

For example, for U.S. English the valid characters are the letters a-Z, the numbers 0-9, a period (.), an underscore (_), a plus sign (+), a hyphen (-), an at sign (@), an ampersand (&), and an asterisk (*). The minimum and maximum lengths of the administrative ID, if there are limits, are imposed by the underlying registry.

-force
Indicates whether or not this task should stop when an error is encountered. The default value is no.

-defer
Indicates whether this task should be run immediately or deferred until the startup of the WebSphere Application Server. The default value is no.

Examples

Interactive mode example usage:

- Using Jacl:

```
$AdminTask unconfigureTAMTAI {-interactive}
```

- Using Jython:

```
AdminTask.unconfigureTAMTAI('-interactive')
```

configureTAMTAIProperties

The configureTAMTAIProperties command adds the custom properties to the security configuration for the embedded Tivoli Access Manager Trust Association Interceptor with classname TAMTrustAsociationInterceptorPlus.

Target object

None.

Required parameters

-loginId
The WebSEAL trusted user is created as discussed in “Creating a trusted user account in Tivoli Access Manager”. See the Configuring single sign-on using trust association interceptor ++ article for more information. The format of the username is the short name representation.

Optional parameters

-checkViaHeader
You can configure TAI so that the via header can be ignored when validating trust for a request. Set this property to false if none of the hosts in the via header need to be trusted. When set to `false` you do not need to set the trusted host names and host ports properties. The only mandatory property to check when via header is false is com.ibm.websphere.security.webseal.loginId. The default value of the check via header property is `false`. When using Tivoli Access Manager plug-in for web servers, set this property to false.

Note: The via header is part of the standard HTTP header that records the server names that the request passed through.

-id
This property indicates a comma-separated list of headers that exists in the request. If all of the configured headers do not exist in the request, trust cannot be established. The default value for the ID property is `iv-creds`. Any other values set in WebSphere Application Server are added to the list along with iv-creds, separated by commas.

-hostnames
Do not set this property if using Tivoli Access Manager plug-in for web servers. The property specifies the host names (case-sensitive) that are both trusted and expected in the request header. Requests arriving from unlisted hosts might not be trusted. If the checkViaHeader property is not set, or is set to `false`, then the trusted host names property has no influence. If the checkViaHeader property is set to `true`, and the trusted host names property is not set, the TAI initialization fails.

-ports
Do not set this property if you are using the Tivoli Access Manager plug-in for web servers. This property is a comma-separated list of trusted host ports. Requests that arrive from unlisted ports might not be trusted. If the checkViaHeader property is not set, or is set to `false`, then this property has no influence. If the checkViaHeader property is set to `true`, and the trusted host ports property is not set in WebSphere Application Server, the TAI initialization fails.

-viaDepth
This property indicates a positive integer that specifies the number of source hosts in the via header to check for trust. By default, every host in the via header is checked, and if any host is not trusted, trust cannot be established. The viaDepth property is used only when some of the hosts in the via header have to be trusted. The setting indicates the number of hosts that are required to be trusted.

As an example, consider the following header:

If in via: HTTP/1.1 webseal1:7002, 1.1 webseal2:7001If the viaDepth property is not set, is set to 2 or is set to 0, and a request with the previous via header is received then both webseal1:7002 and webseal2:7001 need to be trusted. The following configuration then applies:

```
com.ibm.websphere.security.webseal.hostnames = webseal1,webseal2
```

If in com.ibm.websphere.security.webseal.ports = 7002,7001If the viaDepth property is set to 1, and the previous request is received, then only the last host in the via header needs to be trusted. The following configuration then applies:

```
com.ibm.websphere.security.webseal.hostnames = webseal2
com.ibm.websphere.security.webseal.ports = 7001
```

The viaDepth property is set to 0 by default, which means that all of the hosts in the via header are checked for trust.

-ssoPwdExpiry
This property can be used to tell the TAI to ignore proxies as trusted hosts. If set to true, the comments field of the hosts entry in the via header is checked to determine if a host is a proxy. Remember that not all proxies insert comments in the via header indicating that they are proxies. The default value of the ignoreProxy property is `false`. If the checkViaHeader property is set to false, then the ignoreProxy property has no influence in establishing trust.

-viaDepth
This property indicates a positive integer that specifies the number of source hosts in the via header to check for trust. By default, every host in the via header is checked, and if any host is not trusted, trust cannot be established. The viaDepth property is used only when some of the hosts in the via header have to be trusted. The setting indicates the number of hosts that are required to be trusted.

-ssoPwdExpiry
After trust is established for a request, the single sign-on user password is cached, eliminating the need to have the TAI re-authenticate the single sign-on user with Tivoli Access Manager for every request. You can modify the cache timeout period by setting the single sign-on password expiry property to the required time in seconds. If the password expiry property is set to 0, the cached password never expires. The default value for the password expiry property is 600.

-ignoreProxy
This property can be used to tell the TAI to ignore proxies as trusted hosts. If set to `true`, the comments field of the hosts entry in the via header is checked to determine if a host is a proxy. Remember that not all proxies insert comments in the via header indicating that they are proxies. The default value of the ignoreProxy property is `false`. If the checkViaHeader property is set to false, then the ignoreProxy property has no influence in establishing trust.

-configURL
For the TAI to establish trust for a request, it requires that the SvrSslCfg task be run for the Java Virtual Machine on the Application Server and result in the creation of a properties file. If this properties file is not at the default URL, which is file://java.home/PdPerm.properties, the correct URL of the properties file must be set in the configuration URL property. If this property is not set, and the SvrSslCfg-generated properties file is not in the default location, the TAI initialization fails. The default value for the config URL property is `file://${WAS_INSTALL_ROOT}/java/jre/PdPerm.properties`.

Examples

Interactive mode example usage:

- Using Jacl:

```
$AdminTask configureTAMTAIProperties {-interactive}
```

- Using Jython:

```
AdminTask.configureTAMTAIProperties('-interactive')
```

unconfigureTAMTAIProperties

The unconfigureTAMTAIProperties command removes the custom properties from the security configuration for the embedded Tivoli Access Manager Trust Association Interceptor with classname TAMTrustAsociationInterceptorPlus.

Target object

None.

Required parameters

None.

Optional parameters

None.

Examples

Interactive mode example usage:

- Using Jacl:

```
$AdminTask unconfigureTAMTAIProperties {-interactive}
```

- Using Jython:

```
AdminTask.unconfigureTAMTAIProperties('-interactive')
```

configureTAMTAIPdjrte

The configureTAMTAIPdjrte command performs the tasks necessary to fully configure the Tivoli Access Manager Runtime for Java. The specific tasks run are PDJrteCfg and SvrSslCfg.

Target object

None.

Required parameters

-policySvr
This property specifies the name of the Tivoli Access Manager policy server with which the application server communicates. The server is specified by fully qualified host name, the SSL port number, and the rank. The default SSL port number is 7135. For example: *myauth.mycompany.com:7135:1*.

-authSvrs
This property specifies the name of the Tivoli Access Manager authorization server with which the application server communicates. The server is specified by fully-qualified host name, the SSL port number, and the rank. The default SSL port number is 7136. For example: *myauth.mycompany.com:7136:1*. You can specify multiple servers if the entries are separated by a comma (,).

-adminPasswd
This property specifies the password of the Tivoli Access Manager administrator user that is associated with the -adminUid parameter. The password restrictions depend upon the password policy for your Tivoli Access Manager configuration.

Optional parameters

-adminUid
This property specifies the Tivoli Access Manager administrator name. If this option is not specified, `sec_master` is the default. A valid administrative ID is an alphanumeric, case-sensitive string. String values are expected to be characters that are part of the local code set. You cannot use a space in the administrative ID.

For example, for U.S. English. the valid characters are the letters a-Z, the numbers 0-9, a period (.), an underscore (_), a plus sign (+), a hyphen (-), an at sign (@), an ampersand (&), and an asterisk (*). The minimum and maximum lengths of the administrative ID, if there are limits, are imposed by the underlying registry.

-secDomain
This property specifies the Tivoli Access Manager domain name to which the administrator is authenticated. This domain must exist and an administrator ID and password must be valid for this domain. The application server is specified in this domain.

If this property is not specified, the default value is `Default`. The local domain value is retrieved from the configuration file.

A valid domain name is an alphanumeric, case-sensitive string. String values are expected to be characters that are part of the local code set. You cannot use a space in the domain name.

For example, for U.S. English, the valid characters for domain names are the letters a-Z, the numbers 0-9, a period (.), an underscore (_), a plus sign (+), a hyphen (-), an at sign (@), an ampersand (&), and an asterisk (*). The minimum and maximum lengths of the domain name, if there are limits, are imposed by the underlying registry.

-defer
This property indicates whether this task should be run immediately or deferred until the startup of the WebSphere Application Server. The default value is `no`.

Examples

Interactive mode example usage:

- Using Jacl:

```
$AdminTask configureTAMTAIPdjrte {-interactive}
```

- Using Jython:

```
AdminTask.configureTAMTAIPdjrte('-interactive')
```

unconfigureTAMTAIPdjrte

The unconfigureTAMTAIPdjrte command performs the tasks necessary to unconfigure the Tivoli Access Manager Runtime for Java. The specific tasks run are PDJrteCfg and SvrSslCfg.

Target object

None.

Required parameters

-adminPasswd
This property specifies the password of the Tivoli Access Manager administrator user that is associated with the -adminUid parameter. The password restrictions depend upon the password policy for your Tivoli Access Manager configuration.

Optional parameters

-adminUid
This property specifies the Tivoli Access Manager administrator name. If this option is not specified, `sec_master` is the default. A valid administrative ID is an alphanumeric, case-sensitive string. String values are expected to be characters that are part of the local code set. You cannot use a space in the administrative ID.

-force
This property indicates whether or not this task should stop when an error is encountered. The default value is `no`.

-defer
This property indicates whether this task should be run immediately or deferred until the startup of the WebSphere Application Server. The default value is `no`.

Examples

Interactive mode example usage:

- Using Jacl:

```
$AdminTask unconfigureTAMTAIPdjrte {-interactive}
```

- Using Jython:

```
AdminTask.unconfigureTAMTAIPdjrte('-interactive')
```

Disabling embedded Tivoli Access Manager client using the administrative console

To unconfigure the JACC provider for Tivoli Access Manager, you can use the WebSphere Application Server administrative console.

Procedure

1. Click **Security > Global security > External authorization providers**.
2. Make sure that the default option, **Default authorization**, is checked, then click **OK**.
3. On the Global security panel, click **External authorization > External JACC provider**.
4. Under Additional properties, click **Tivoli Access Manager Properties**. The configuration screen for the JACC provider for Tivoli Access Manager is displayed.
5. Clear the **Enable embedded Tivoli Access Manager** option. If you want to ignore errors when unconfiguring, select the **Ignore errors during embedded Tivoli Access Manager disablement** option. Select this option only when the Tivoli Access Manager domain is in an irreparable state.
6. Click **OK**.
7. Optional: If you want security enabled without Tivoli Access Manager re-enable administrative security.
8. Restart all WebSphere Application Server instances for the changes to take effect.

Forcing the unconfiguration of the Tivoli Access Manager JACC provider

If you find you cannot restart WebSphere Application Server after configuring the JACC provider for Tivoli Access Manager a utility is available to clear the security configuration and return WebSphere Application Server to an operable state.

About this task

The utility removes all of the PDLoginModuleWrapper entries as well as the Tivoli Access Manager authorization table from `security.xml` and `wsjaas.conf` files. This utility effectively removes the JACC provider for Tivoli Access Manager.

Procedure

1. Back up the `security.xml` and `wsjaas.conf` files.
2. Enter the following command as one continuous line.

```
app_server_root/java/jre/bin/java
-classpath "app_server_root /$WAS_HOME/plug-in/com.ibm.ws.runtime_1.0.0.jar"
 com.tivoli.pd.as.jacc.cfg.CleanSecXML
 fully_qualified_path/security.xml fully_qualified_path/wsjaas.conf
```

Propagating security policies and roles for previously deployed applications

Use this task to propagate security policies and roles to the external Java Authorization Contract for Containers (JACC) provider.

Before you begin

The external JACC provider must be configured before following these steps.

About this task

After switching to use the external JACC provider you can follow these steps to avoid having to redeploy your existing applications. Updating using these steps retrieves the security policy and roles from the deployed applications and propagates it to the external JACC provider removing the need for the applications to be redeployed.

Procedure

1. From the WebSphere Application Server administrative console, click **Security** > **Global security** > **External authorization providers**.
2. Select the appropriate security policy and role updating option.
   - Select **Don't update provider** to not propagate any security policies or roles
   - Select **Update with all applications** to propagate security policies and roles for all applications
   - Select **Update with application names listed** to propagate security policies and roles for the selected applications. If multiple applications should be updated, separate the application names with commas.
3. Click **Apply**.

Results

After completing this task your security policies and roles have been successfully propagated to the external JACC provider.

Authorizing access to administrative roles

You can assign users and groups to administrative roles to identify users who can perform WebSphere Application Server administrative functions.

Before you begin

Administrative roles enable you to control access to WebSphere Application Server administrative functions. Refer to the descriptions of these roles in Administrative roles.

- Before you assign users to administrative roles, you must set up your user registry. For information on the supported registry types, see “Selecting a registry or repository” on page 159.
- The following steps are needed to assign users to administrative roles.

About this task

You use the administrative console to assign users and groups to administrative roles and to identify users who can perform WebSphere Application Server administrative functions. In the administrative console,

Procedure

1. Click **Users and Groups**. Click either **Administrative User Roles** or **Administrative Group Roles**.
2. To add a user or a group, click **Add** on the Console users or Console groups panel.
3. To add a new administrator user, follow the instructions on the page to specify a user, and select the **Administrator** role. Once the user is added to the Mapped to role list, click **OK**. The specified user is mapped to the security role.
4. To add a new administrative group, follow the instructions on the page to specify either a group name or a Special subject, highlight the **Administrator** role, and click **OK**. The specified group or special subject is mapped to the security role.
5. To remove a user or group assignment, click **Remove** on the Console Users or the Console Groups panel. On the Console Users or the Console Groups panel, select the check box of the user or group to remove and click **OK**.
6. To manage the set of users or groups to display, click **Show filter function** on the User Roles or Group Roles panel. In the **Search term(s)** box, type a value, then click **Go**. For example, user* displays only users with the user prefix.
7. After the modifications are complete, click **Save** to save the mappings.
8. Restart the application server for changes to take effect.

What to do next

After you assign users to administrative roles, you must restart the server for the new roles to take effect. However, the administrative resources are not protected until you enable security.

Administrative user roles settings and CORBA naming service user settings

Use the Administrative User Roles page to give users specific authority to administer application servers through tools such as the administrative console or wsadmin scripting. The authority requirements are only effective when global security is enabled. Use the Common Object Request Broker Architecture (CORBA) naming service users settings page to manage CORBA naming service users settings.

To view the Console Users administrative console page, complete either of the following steps:

- Click **Security > Global security > Administrative User Roles**.
- Click **Users and Groups > Administrative User Roles**.

Note: If you are using local OS, the SIB administrative security panel's searches can use both the "?" and "*" search characters. However. if you switch to federated repositories, the searches will not work with the "?" character but will with the "*" character.

To view the CORBA naming service groups administrative console page, click **Environment > Naming > CORBA Naming Service Groups**.

Click **Refresh All** to automatically update the node agent and all of the nodes when a new user is created with the Administrator or Admin Security Manager role. When you click **Refresh All**, you do not need to manually restart the node agent under an existing Administrator before the new user is recognized with one of these roles. This button automatically invokes the AuthorizationManager refreshAll MBean method. To invoke this method manually, read about Fine-grained administrative security in heterogeneous and single-server environments.

User (Administrative user roles)

Specifies users.

The users that are entered must exist in the configured active user registry.

Information	Value
Data type:	String

User (CORBA naming service users)

Specifies CORBA naming service users.

The users that are entered must exist in the configured active user registry.

Information	Value
Data type:	String

Role (Administrative user roles)

Specifies user roles.

The following administrative roles provide different degrees of authority that are needed to perform certain application server administrative functions:

Administrator

The administrator role has operator permissions, configurator permissions, and the permission that

is required to access sensitive data including server password, Lightweight Third Party Authentication (LTPA) password and keys, and so on.

Operator
The operator role has monitor permissions and can change the run-time state. For example, the operator can start or stop services.

Configurator
The configurator role has monitor permissions and can change the WebSphere Application Server configuration.

Deployer
The deployer role can complete both configuration actions and run-time operations on applications.

Monitor
The monitor role has the least permissions. This role primarily confines the user to viewing the application server configuration and current state.

adminsecuritymanager
The adminsecuritymanager role has privileges for managing users and groups from within the administrative console and determines who has access to modify users and groups using administrative role mapping. Only the adminsecuritymanager role can map users and groups to administrative roles, and by default, AdminId is granted to the adminsecuritymanager.

iscadmins
The iscadmins role has administrator privileges for managing users and groups from within the administrative console only.

Note: To manage users and groups, click **Users and Groups** in the console navigation tree. Click either **Manage Users** or **Manage Groups**.

Information	Value
Data type:	String
Range:	Administrator, Operator, Configurator, Deployer, Monitor, and iscadmins

Note: Other arbitrary administrative roles might also be visible in the administrative console collection table. Other contributors to the console might create these additional roles, which can be used for applications that are deployed to the console.

Role (CORBA naming service users)

Specifies naming service user roles.

A number of naming roles are defined to provide degrees of authority that are needed to perform certain application server naming service functions. The authorization policy is only enforced when global security is enabled. The following roles are valid: CosNamingRead, CosNamingWrite, CosNamingCreate, and CosNamingDelete.

The roles now have authority levels from low to high:

CosNamingRead
You can query the application server name space by using, for example, the Java Naming and Directory Interface (JNDI) lookup method. The EVERYONE special-subject is the default policy for this role.

CosNamingWrite
You can perform write operations such as JNDI bind, rebind, or unbind, plus CosNamingRead operations.

CosNamingCreate
You can create new objects in the name space through operations such as JNDI createSubcontext and CosNamingWrite operations.

CosNamingDelete

You can destroy objects in the name space, for example using the JNDI destroySubcontext method and CosNamingCreate operations.

Information	Value
Data type:	String
Range:	CosNamingRead, CosNamingWrite, CosNamingCreate and CosNamingDelete

Login status (Administrative user roles)

Specifies whether the user is active or inactive.

Administrative group roles and CORBA naming service groups

Use the Administrative Group Roles page to give groups specific authority to administer application servers through tools such as the administrative console or wsadmin scripting. The authority requirements are only effective when administrative security is enabled. Use the Common Object Request Broker Architecture (CORBA) naming service groups page to manage CORBA Naming Service groups settings.

To view the Console Groups administrative console page, complete either of the following steps:

- Click **Security > Global security > Administrative Group Roles**.
- Click **Users and Groups > Administrative Group Roles**.

To view the CORBA naming service groups administrative console page, click **Environment > Naming > CORBA Naming Service Groups**.

Click **Refresh All** to automatically update the node agent and all of the nodes when a new user is created with the Administrator or Admin Security Manager role. When you click **Refresh All**, you do not need to manually restart the node agent under an existing Administrator before the new user is recognized with one of these roles. This button automatically invokes the AuthorizationManager refreshAll MBean method. To invoke this method manually, read about Fine-grained administrative security in heterogeneous and single-server environments.

Group (CORBA naming service groups)

Identifies CORBA naming service groups.

In previous releases of WebSphere Application Server, there were two default groups: ALL AUTHENTICATED and EVERYONE. However, EVERYONE is now the only default group, and it provides CosNamingRead privileges only.

Information	Value
Data type:	String
Range:	EVERYONE

Role (CORBA naming service groups)

Identifies naming service group roles.

A number of naming roles are defined to provide the degrees of authority that are needed to perform certain application server naming service functions. The authorization policy is only enforced when global security is enabled.

Four name space security roles are available: CosNamingRead, CosNamingWrite, CosNamingCreate, and CosNamingDelete. The roles have authority levels from low to high:

Cos Naming Read
You can query the application server name space using, for example, the Java Naming and Directory Interface (JNDI) lookup method. The EVERYONE special-subject is the default policy for this role.

Cos Naming Write
You can perform write operations such as JNDI bind, rebind, or unbind, and CosNamingRead operations. The ALL_AUTHENTICATED special-subject is the default policy for this role.

Cos Naming Create
You can create new objects in the name space through operations such as JNDI createSubcontext and CosNamingWrite operations. The ALL_AUTHENTICATED special-subject is the default policy for this role.

Cos Naming Delete
You can destroy objects in the name space, for example using the JNDI destroySubcontext method and CosNamingCreate operations. The ALL_AUTHENTICATED special-subject is the default policy for this role.

Information	Value
Data type:	String
Range:	CosNamingRead, CosNamingWrite, CosNamingCreate, and CosNamingDelete

Group (Administrative group roles)
Specifies groups.

The ALL_AUTHENTICATED and the EVERYONE groups can have the following role privileges: Administrator, Configurator, Operator, and Monitor.

Information	Value
Data type:	String
Range:	ALL_AUTHENTICATED, EVERYONE

Role (Administrative group roles)
Specifies user roles.

The following administrative roles provide different degrees of authority needed to perform certain application server administrative functions:

Administrator
The administrator role has operator permissions, configurator permissions, and the permission that is required to access sensitive data, including server password, Lightweight Third Party Authentication (LTPA) password and keys, and so on.

Operator
The operator role has monitor permissions and can change the run-time state. For example, the operator can start or stop services.

Configurator
The configurator role has monitor permissions and can change the application server configuration.

Deployer
The deployer role can perform both configuration actions and runtime operations on applications.

Monitor
The monitor role has the least permissions. This role primarily confines the user to viewing the application server configuration and current state.

iscadmins
The iscadmins role has administrator privileges for managing users and groups from within the administrative console only.

Note: To manage users and groups, click **Users and Groups** in the console navigation tree. Click either **Manage Users** or **Manage Groups**.

Auditor

The auditor can view and modify the configuration settings for the security auditing subsystem. The auditor role includes the monitor role.

Information	Value
Data type:	String
Range:	Administrator, Operator, Configurator, Monitor, Deployer and iscadmins

Note: Other arbitrary administrative roles might also be visible in the administrative console collection table. Other contributors to the console might create these additional roles, which can be used for applications that are deployed to the console.

Assigning users to naming roles

Use this task to assign users to naming roles by using the administrative console.

About this task

The following steps are needed to assign users to naming roles. In the administrative console, click **Environment > Naming**, and click **CORBA Naming Service Users** or **CORBA Naming Service Groups**.

Procedure

1. Click **Add** on the CORBA Naming Service Users or the CORBA Naming Service Groups panel.
2. To add a new naming service user, follow the instructions on the page to specify a user, and select one or more roles. Once the user is added to the "Mapped to role" list, click **OK**. The specified user is mapped to one or more security roles.
3. To add a new naming service group, follow the instructions on the page to specify either a group name or a Special subject, highlight one or more roles, and click **OK**. The specified group or special subject are mapped to one or more the security roles
4. To remove a user or group assignment, go to the **CORBA Naming Service Users** or **CORBA Naming Service Groups** panel. Select the check box next to the user or group that you want to remove and click **Remove**.
5. To manage the set of users or groups to display, expand the **Filter** folder on the right panel, and modify the filter text box. For example, setting the filter to `user*` displays only users with the `user` prefix.
6. After modifications are complete, click **Save** to save the mappings. Restart the server for the changes to take effect.

Example

The default naming security policy is to grant all users read access to the CosNaming space and to grant any valid user the privilege to modify the contents of the CosNaming space. You can perform the previously mentioned steps to restrict user access to the CosNaming space. However, use caution when changing the naming security policy. Unless a Java Platform, Enterprise Edition (Java EE) application has clearly specified its naming space access requirements, changing the default policy can result in unexpected org.omg.CORBA.NO_PERMISSION exceptions at runtime.

Propagating administrative role changes to Tivoli Access Manager

These steps provide an example of how to migrate the `admin-authz.xml` file.

About this task

Additions and changes to console users and groups are not automatically added to the Tivoli Access Manager object space after the Java Authorization Contract for Containers (JACC) provider for Tivoli Access Manager is configured. Changes to console users and groups are saved in the `admin-authz.xml` file and this file must be migrated before any changes take effect. The JACC provider for Tivoli Access Manager includes the **migrateEAR** migration utility for incorporating console user and group changes into the Tivoli Access Manager object space.

Note: The **migrateEAR** utility is used to migrate the changes made to console users and groups after the JACC provider for Tivoli Access Manager is configured. The utility does not need to run for changes and additions to console users and groups made prior to the configuration of the JACC provider for Tivoli Access Manager because the changes made to the `admin-authz.xml` and `naming-authz.xml` files are automatically migrated at configuration time. Furthermore, the migration tool does not need to run before deploying standard Java Platform, Enterprise Edition (Java EE) applications; Java EE application policy deployment is also performed automatically.

For example, if you wanted to migrate the `admin-authz.xml` file, perform the following steps:

Procedure

1. Set up the environment.

   Before running the `migrateEAR` utility, set up the environment by running the `setupCmdLine.bat` or `setupCmdLine.sh` file that is located in the *app_server_root*`/bin` directory.

   Make sure that the *WAS_HOME* environment variable is set to the WebSphere Application Server installation directory.
2. Change to the *app_server_root*`/bin` directory where the `migrateEAR` utility is located.
3. Run the **migrateEAR** utility to migrate the data contained in the `admin-authz.xml` file. Use the parameter descriptions that are listed in "migrateEAR utility for Tivoli Access Manager."

   AIX HP-UX Solaris For example:

```
migrateEAR
-j "app_server_root/profiles/profile_name/config/cells/cell_name/xml_filename"
-a sec_master
-p password
-w wsadmin
-d o=ibm,c=us
-c file:/"app_server_root/java/jre/PdPerm.properties"
-z Roles
```

   where *xml_filename* might be `admin-authz.xml` or `naming-authz.xml`. The `-z Roles` parameter is optional and when specified adds a subdirectory under the current directory structure in which to store the role mapping. For example,

```
/WebAppServer/deployedResouces/Roles
```

   If `-z Roles` is not specified, the role mapping is stored in the current directory structure. For example,

```
/WebAppServer/deployedResouces
```

   A status message is displayed when the migration completes. Output of the utility is logged to the `pdwas_migrate.log` file, which is created in the directory where the utility is run. Check the log file after each migration. If the log file displays errors, check the last recorded transaction, correct the source of the error, and rerun the migration utility. If the migration is unsuccessful, verify that you supplied the correct values for the `-c` and `-j` options.
4. WebSphere Application Server does not require a restart for the changes to take effect.

migrateEAR utility for Tivoli Access Manager

The **migrateEAR** utility migrates changes made to console users and groups in the `admin-authz.xml` and `naming-authz.xml` files into the Tivoli Access Manager object space.

Syntax

```
migrateEAR
-j fully_qualified_filename
-c pdPerm.properties_file_location
-a Tivoli_Access_Manager_administrator_ID
-p Tivoli_Access_Manager_administrator_password
-w WebSphere_Application_Server_administrator_user_name
-d user_registry_domain_suffix
[-r root_objectspace_name]
[-t ssl_timeout]
[-z role_mapping_location]
```

Parameters

Attention: Windows In the following parameters, use the absolute path instead of a variable.

-a*Tivoli_Access_Manager_administrator_ID*
The administrative user identifier. The administrative user must have the privileges required to create users, objects, and access control lists (ACLs). For example, `-a sec_master`.

This parameter is optional. When the parameter is not specified, you are prompted to supply it at run time.

-c *PdPerm.properties_file_location*
The Uniform Resource Indicator (URI) location of the `PdPerm.properties` file that is configured by the pdwascfg utility. When WebSphere Application Server is installed in the default location, the URI is:

Solaris Linux HP-UX

```
file:/opt/IBM/WebSphere/AppServer/java/jre/PdPerm.properties
```

AIX

```
file:/usr/IBM/WebSphere/AppServer/java/jre/PdPerm.properties
```

Windows

```
file:/"C:/Program Files/IBM/WebSphere/AppServer/java/jre/PdPerm.properties"
```

-d *user_registry_domain_suffix*
The domain suffix for the user registry to use. For example, for Lightweight Directory Access Protocol (LDAP) user registries, this value is the domain suffix, such as: `"o=ibm,c=us"`

Windows Windows platforms require that the domain suffix is enclosed within quotes.

You can use the **pdadmin user show** command to display the distinguished name (DN) for a user.

-j *fully_qualified_pathname*
The fully qualified path and file name of the Java 2 Platform, Enterprise Edition application archive file ,`admin-authz.xml` or the roles definitions file `naming-authz.xml` that is used for a naming operation authorization. Optionally, this path can also be a directory of an expanded enterprise application. For example, when WebSphere Application Server is installed in the default location, the path to the data files to migrate includes:

Solaris Linux HP-UX

```
file:/opt/IBM/WebSphere/AppServer/profiles/profile_name/config/cells
/cell_name/admin-authz.xml
```

AIX

```
file:/usr/IBM/WebSphere/AppServer/profiles/profile_name/config/cells
/cell_name/admin-authz.xml
```

Windows

```
"C:/Program Files/IBM/WebSphere/AppServer/profiles/profile_name/config/cells
/cell_name/admin-authz.xml"
```

-p *Tivoli_Access_Manager_administrator_password*
The password for the Tivoli Access Manager administrative user. The administrative user must have

the privileges that are required to create users, objects, and access control lists (ACLs). For example, you can specify the password for the `-a sec_master` administrative user as `-p myPassword`.

When this parameter is not specified, the user is prompted to supply the password for the administrative user name.

-r *root_objectspace_name*
The space name of the root object. The value is the name of the root of the protected object namespace hierarchy that is created for WebSphere Application Server policy data.

The default value for the root object space is `WebAppServer`.

Set the Tivoli Access Manager root object space name by modifying the `amwas.amjacc.template.properties` file prior to configuring the Java Authorization Contract for Containers (JACC) provider for Tivoli Access Manager for the first time. Use this option if the default object space value is not used in the configuration of the Tivoli Access Manager JACC provider for Tivoli Access Manager.

Do not change the Tivoli Access Manager object space name after the Tivoli Access Manager JACC provider is configured.

-t *ssl_timeout*
The number of minutes for the Secure Sockets Layer (SSL) timeout. This parameter is used to disconnect and reconnect the SSL context between the Tivoli Access Manager authorization server and the policy server before the default connection times out.

The default is `60` minutes. The minimum value is `10` minutes. The maximum value cannot exceed the Tivoli Access Manager ssl-v3-timeout value. The default value for ssl-v3-timeout is `120` minutes.

If you are not familiar with the administration of this value, you can safely use the default value.

-w *WebSphere_Application_Server_administrator_user_name*
The user name that is configured in the WebSphere Application Server security user registry field as the administrator. This value matches the account that you created or imported in “Creating the security administrative user for Tivoli Access Manager” on page 605. Access permission for this user is needed to create or update the Tivoli Access Manager protected object space.

When the WebSphere Application Server administrative user does not already exist in the protected object space, it is created or imported. In this case, a random password is generated for the user and the account is set to `not valid`. Change this password to a known value and set the account to `valid`.

A protected object and access control list (ACL) are created. The administrative user is added to the pdwas-admin group with the following ACL attributes:

T Traverse permission

i Invoke permission

WebAppServer
You can overwrite the action group name. The default name is `WebAppServer`. This action group name and the matching root object space can be overwritten when the migration utility is run with the **-r** option.

-z *role_mapping_location*
The location where the role mapping is to be stored when migrating administration applications. The default location is to place the role mapping in the current directory structure, such as:

```
/WebAppServer/deployedResouces
```

Specifying the -z option adds another directory level in which to store the role mapping. For example, if you specify `-z Roles` in the migrateEAR utility, the role mapping is stored in the directory structure as follows:

```
/WebAppServer/deployedResouces/Roles
```

Comments

This utility migrates security policy information from deployment descriptors or enterprise archive files to Tivoli Access Manager for WebSphere Application Server. The script calls com.tivoli.pdwas.migrate.Migrate the Java class.

Before invoking the script you must run the **setupCmdLine.bat** or the **setupCmdLine.sh** commands. These files can be found in the `%WAS_HOME%/bin` directory.

The script is dependent on finding the correct environment variables for the location of prerequisite software.

The script calls Java code with the following options:

-Dpdwas.lang.home
The directory that contains the native language support libraries that are provided with the JACC provider for Tivoli Access Manager. These libraries are located in a subdirectory under the JACC provider for Tivoli Access Manager installation directory. For example: `-Dpdwas.lang.home=%PDWAS_HOME%\java\nls`

-cp %CLASSPATH% com.tivoli.pdwas.migrate.Migrate
The CLASSPATH variable must be set correctly for your Java installation.

Windows Both the -j option and the -c option can reference the %WAS_HOME% variable to determine where WebSphere Application Server is installed. This information is used to:

- Build the full path name of the enterprise archive file.
- Build the full URI path name to the location of the `PdPerm.properties` file.

To enable a new user access to the administrative group in WebSphere Application Server, it is recommended that the user be added to the pdwas-admin group after JACC has been enabled. You can enter the administrative primary ID (adminID) in the group. This is required when the serverID is not the same as the adminID.

The following is an example of this command:

```
pdadmin> group modify pdwas-admin add adminID
```

Return codes

The utility can return the following exit status codes:

0 The command completed successfully.

1 The command failed.

Assigning users from a foreign realm to the admin-authz.xml

Operating with the administrative agent and job manager topology allows more situations where you might need to add an administrative user from a different registry into your administrative authorization table (admin-authz.xml). Each administrative user that needs to be added requires the "accessID" format of the user from the remote registry. When that user finally is active in the local cell, the authorization table will already have that accessID that is required. This task demonstrates how this assignment of users is performed.

Procedure

1. You need to determine the accessId for a user on the remote registry. To do this, you call the following wsadmin task and query based on a user filter. The following example illustrates a query from the registry realm "BIRKT60" with a userFilter of "localuser*". This query returns any user from this realm

that begins with "localuser". The resulting accessId is the one you need to specify in the target administrative authorization table in the following step. Connect to the sending administrative process:

```
wsadmin> $AdminTask listRegistryUsers {-securityRealmName BIRKT60 -displayAccessIds true -userFilter localuser*}
{name BIRKT60\localuser@BIRKT60}
{accessId user:BIRKT60/S-1-5-21-3033296400-14683092-2821094880-1007}
```

2. Add "localuser" to the target `admin-authz.xml` using the following wsadmin task. Connected to the receiving administrative process:

```
wsadmin> $AdminTask mapUsersToAdminRole {-roleName administrator -userids {localuser }
-accessids {user:BIRKT60/S-1-5-21-3033296400-14683092-2821094880-1007 }}
```

3. Save the changes.

Results

This task updates the `admin-authz.xml` in the receiving administrative process to allow a "cross-realm authorization" to succeed. The example illustrated here was for a LocalOS registry user. Performing the same task for an LDAP accessId produces results that look more like a realm and distinguished name (DN).

Note: If you change your realm you must repeat this process with the new realm name.

Fine-grained administrative security

In releases prior to WebSphere Application Server version 6.1, users granted administrative roles could administer all of the resources under the cell. WebSphere Application Server is now more fine-grained, meaning that access can be granted to each user per resource.

For example, users can be granted configurator access to a specific instance of a resource only (an application, an application server or a node). Users cannot access any other resources outside of the resources assigned to them. The administrative roles are now per resource rather than to the entire cell. However, there is a cell-wide authorization group for backward compatibility. Users assigned to administrative roles in the cell-wide authorization group can still access all of the resources within the cell.

Note: Nodes prior to WebSphere Application Server Version 6.1 in a mixed cell environment are filtered out of resource mapping.

To achieve this instance-based security or fine-grained security, resources that require the same privileges are placed in a group called the administrative authorization group or authorization group. Users can be granted access to the authorization group by assigning to them the required administrative role.

Fine-grained administrative security can also be used in single-server environments. Various applications in the single server can be grouped and placed in different authorization groups. Therefore, there are different authorization constraints for different applications. Note that the server itself cannot be part of any authorization group in a single-server environment.

You can assign users and groups to the adminsecuritymanager role on the cell level through wsadmin scripts and the administrative console. Using the adminsecuritymanager role, you can assign users and groups to the administrative user roles and administrative group roles.

When fine grained administrative security is used, users granted the adminsecuritymanager role can manage authorization groups. See “Administrative roles and naming service authorization” on page 566 for detailed explanations of all administrative roles.

An administrator cannot assign users and groups to the administrative user roles and administrative group roles, including the adminsecuritymanager role. See “Administrative roles” on page 574 for more details.

There are several administrative security commands that can be used to create authorization groups, map resources to authorization groups, and to assign users to administrative roles within the authorization groups. Following are some examples using wsadmin:

- Create a new authorization group:

```
$AdminTask createAuthorizationGroup {-authorizationGroupName authGroup1}
```

- Deleting an authorization group:

```
$AdminTask deleteAuthorizationGroup {-authorizationGroupName groupName}
```

- Add resources to an authorization group:

```
$AdminTask addResourceToAuthorizationGroup
{-authorizationGroupName groupName -resourceName Application=app1}
```

- Remove resources from an authorization group:

```
$AdminTask removeResourceFromAuthorizationGroup
{-authorizationGroupName groupName -resourceName Application=app1}
```

- Add user IDs to roles in an authorization group:

```
$AdminTask mapUsersToAdminRole {-authorizationGroupName groupName
-roleName administrator -userids user1}
```

- Add group IDs to roles in an authorization group:

```
$AdminTask mapGroupsToAdminRole {-authorizationGroupName groupName
-roleName administrator -groupids group1}
```

- Remove user IDs from roles in an authorization group:

```
AdminTask removeUsersFromAdminRole {-authorizationGroupName
groupName -roleName administrator -userids user1}
```

- Remove group IDs from roles in an authorization group:

```
$AdminTask removeGroupsFromAdminRole {-authorizationGroupName
groupName -roleName administrator -groupids group1}
```

Resources that can be added to an authorization group

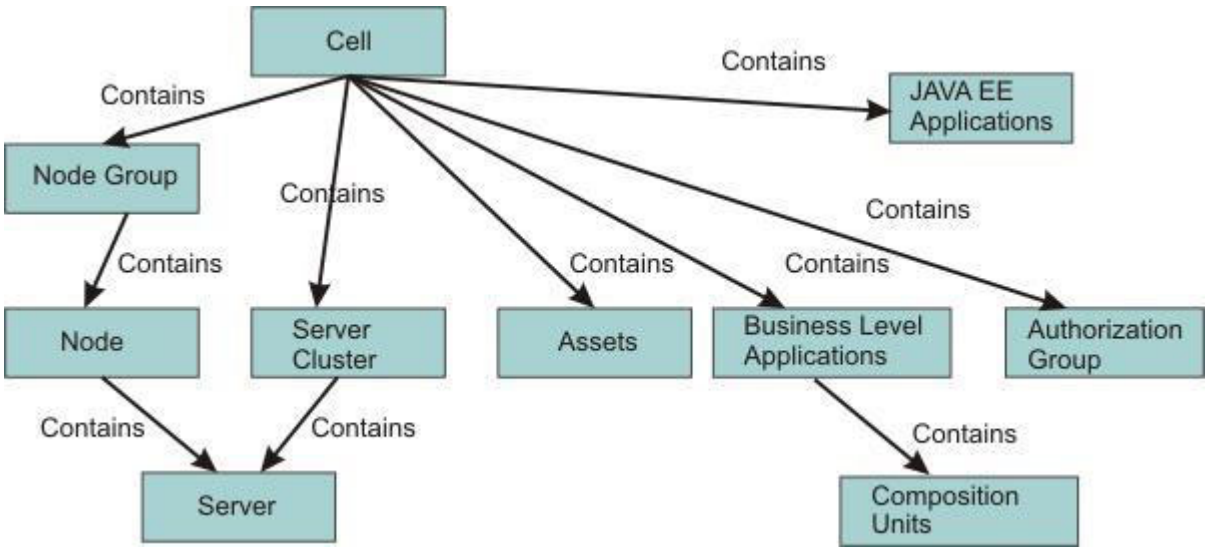
You can add only resources of the following types to an authorization group:

- Cell
- Node
- ServerCluster
- Server
- Application
- NodeGroup

If a resource is not one of the types previously listed, its parent resource will be used.

A resource can only belong to one authorization group. However, there is a containment relationship among resources. If a parent resource belongs to a different authorization group than that of its child resource, the child resource implicitly will belong to multiple authorization groups. You cannot add the same resource to more than one authorization group.

The following diagram shows the containment relationship among resources:



The privileges required for actions on resources depends on two factors:

- The authorization group of the administrative resource. If a user is granted access to an authorization group, all of the resources in that group will be included.
- The containment relationship of the resource. If a user is granted access to a parent resource, all of the children resources will be included.

Keystore management requires a user to have cell-level administrative privileges because they are created and managed at the cell level. Fine-grained security access to a specific resource does not allow management of the associated keystores.

Table 83. Privileges required to access various administrative resources. The privileges required to access various administrative resources are shown in the following table:

Resource	Action	Required roles
Server	Start, stop, runtime operations	Server-operator, node-operator, cell-operator
Server	New, delete	Node-configurator, cell-configurator
Server	Edit configuration	Server-configurator, node-configurator, cell-configurator
Server	View configuration, runtime status	Server-monitor, node-monitor, cell-monitor
Node	Restart, stop, sync	Node-operator, Cell-operator
Node	Add, delete	Cell-configurator
Node	Edit configuration	Node-configurator, cell-configurator
Node	View configuration, runtime status	Node-monitor, cell-monitor
Cluster	Start, stop, runtime operations	Cluster-operator, cell-operator
Cluster	New, delete	Cell-configurator
Cluster	Edit configuration	Cluster-configurator, cell-configurator
Cluster	View configuration, runtime status	Cluster-monitor, cell-monitor
Cluster member	Start, stop, runtime operations	Server-operator, cluster-operator, node-operator, cell-operator
Cluster member	New, delete	Node-configurator, cell-configurator
Cluster member	Edit configuration	Server-configurator, cluster-configurator, node-configurator, cell-configurator

Table 83. Privileges required to access various administrative resources (continued). The privileges required to access various administrative resources are shown in the following table:

Resource	Action	Required roles
Cluster member	View configuration, runtime status	Server-monitor, cluster-monitor, node-monitor, cell-monitor
Application	All operations	Refer to the section "Deployer roles" in “Administrative roles” on page 574.
Node, cluster	Add, delete	Cell-configurator

The server-operator role is the operator role of the authorization group to which the server instance is part of. Similarly, the node-operator role is in the operator role of the authorization group to which the node instance is part of.

To use fine-grained administrative security in the administrative console, a user should be granted a monitor role at the cell level at minimum. However, to login using wsadmin, a user should be granted a monitor role for any authorization group.

Example: Using fine-grained security.

The following scenarios describe the use of fine-grained administrative security, particularly the new deployment role.

Deployment role scenario 1.

In the following scenario, there are four applications configured on server S1, as shown in the following table. Each application must be isolated so that the administrator of one application cannot modify another application. Assume that only user1 can manage application A1, user2 can manage applications A2 and A3, and only user3 can manage application A4.

Note: It is not recommended to have an application in one group and its target server in another group. However, that is not always possible. It is common to have many applications on one server. It is still sometimes necessary to isolate the administration of applications running on the same server.

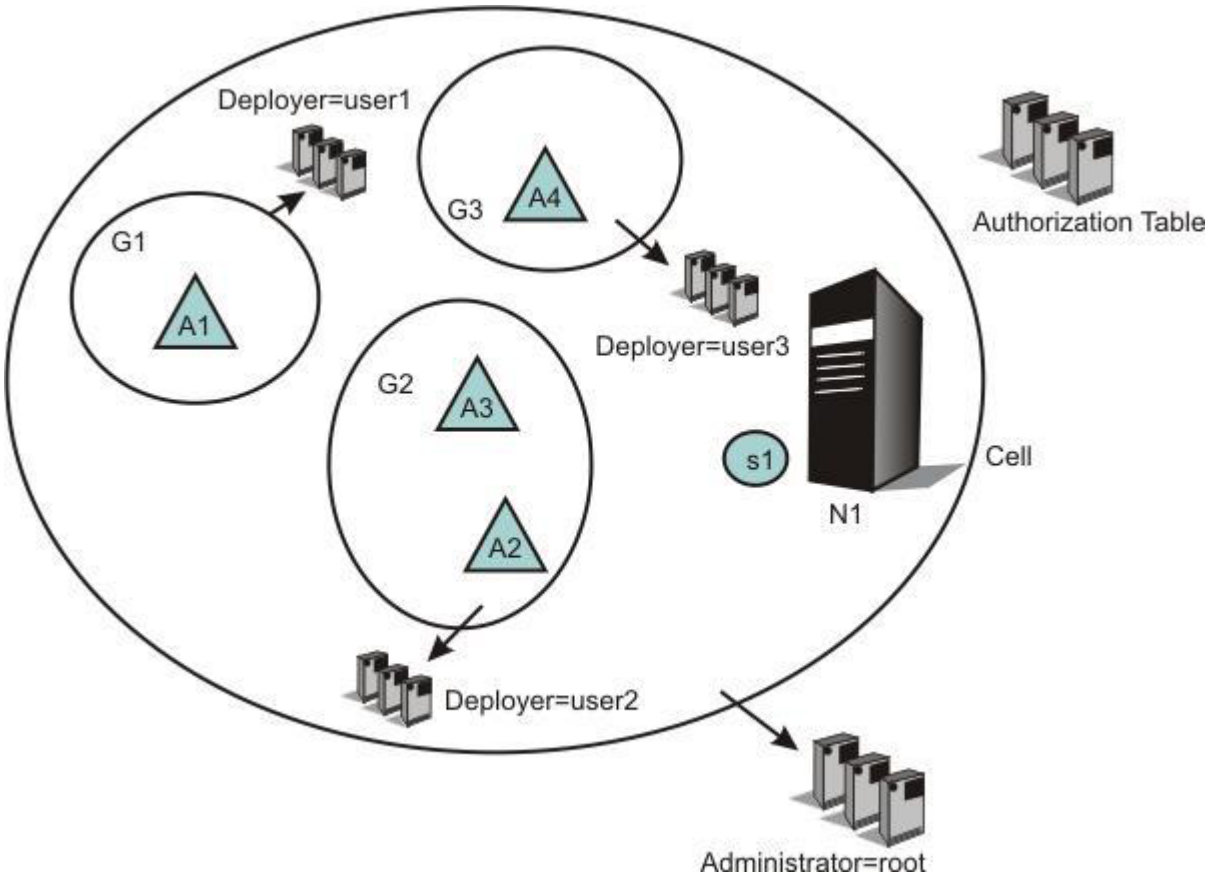
One example is an Application Service Provider (ASP), where a single application server can have multiple vendor applications. In this instance the server administrator is responsible for installing all of the vendor applications. Once applications are installed, each vendor can manage their own application without interfering with other vendor's applications.

Table 84. Deployment role scenario 1 applications.

This table lists the Deployment role scenario 1 applications.

Application	Server	Node
A1	S1	N1
A2	S1	N1
A3	S1	N1
A4	S1	N1

We can configure authorization groups as shown in the following diagram:



In the diagram, application A1 is in authorization group G1, applications A2 and A3 are in authorization group G2, and application A4 is in authorization group G3.

A deployer role is assigned from authorization group G1 to user1, from authorization group G2 to user2, and from authorization group G3 to user3.

Consequently, user1 can perform all of the operations on application A1, user2 on applications A2 and A3, and user3 on application A4. Since all applications share the same server, we cannot put the same server on all authorization groups. Only a cell-level administrator can install an application. After the installation of an application is complete, the deployer of each application can modify their own. To start and stop the server, cell-level administrative authority is required. This type of scenario is useful in an ASP environment.

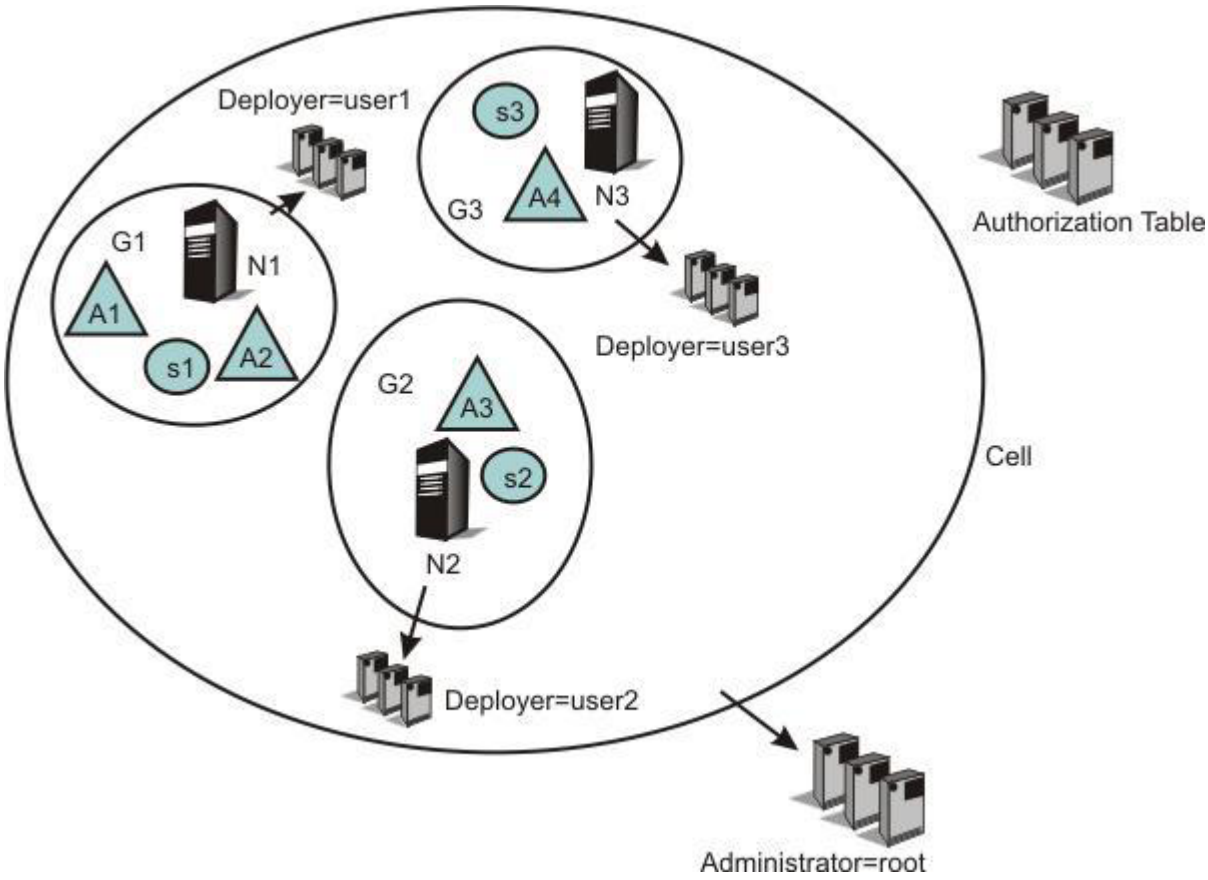
Deployment role scenario 2.

In the following scenario, a group of applications require the same administrative roles to one server. In this example, applications A1 and A2 are related applications, and can be administrated by one set of administrators. They are running on the same server (S1). Applications A3 and A4 require a different set of administrators, and are running on servers S2 and S3 respectively.

Table 85. Deployment role scenario 2 applications.

This table lists the Deployment role scenario 2 applications.

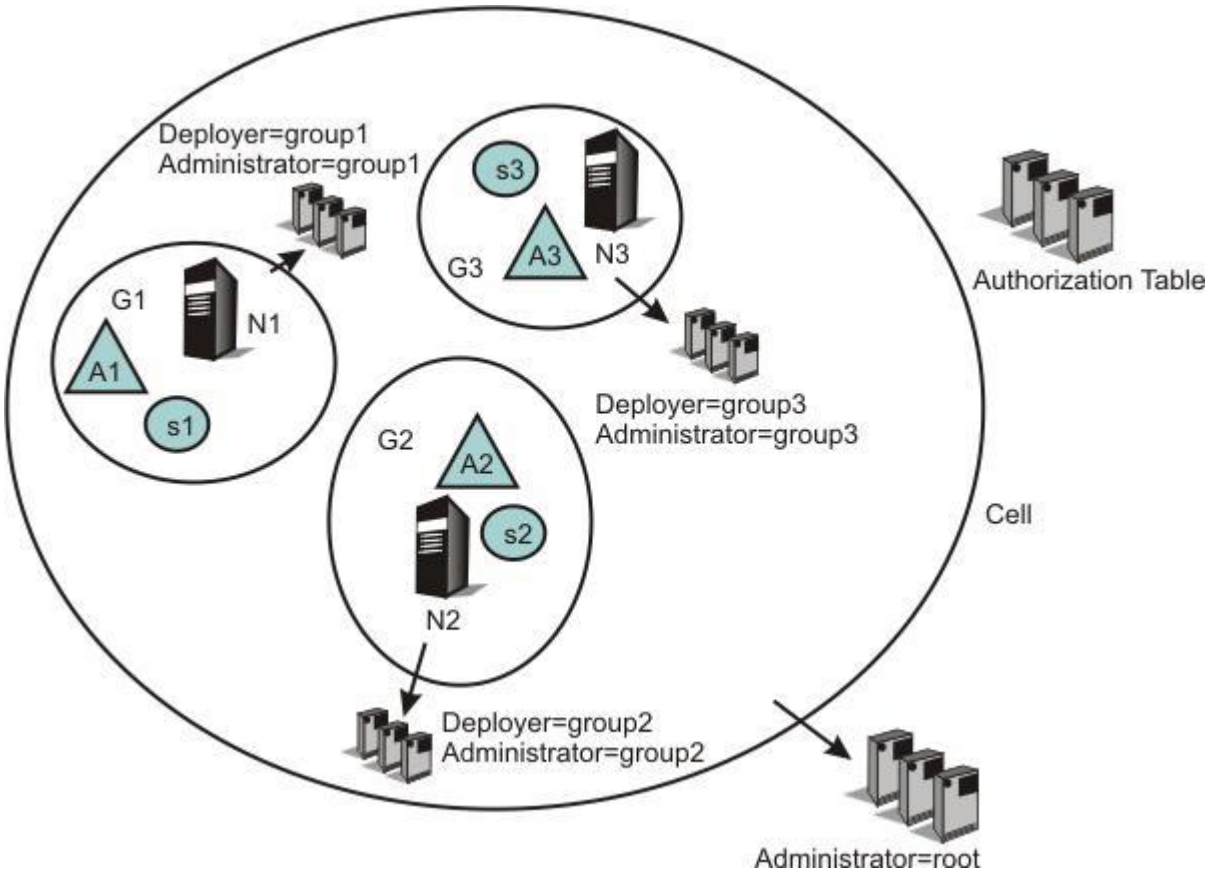
Application	Server	Node
A1	S1	N1
A2	S1	N1
A3	S2	N2
A4	S3	N3



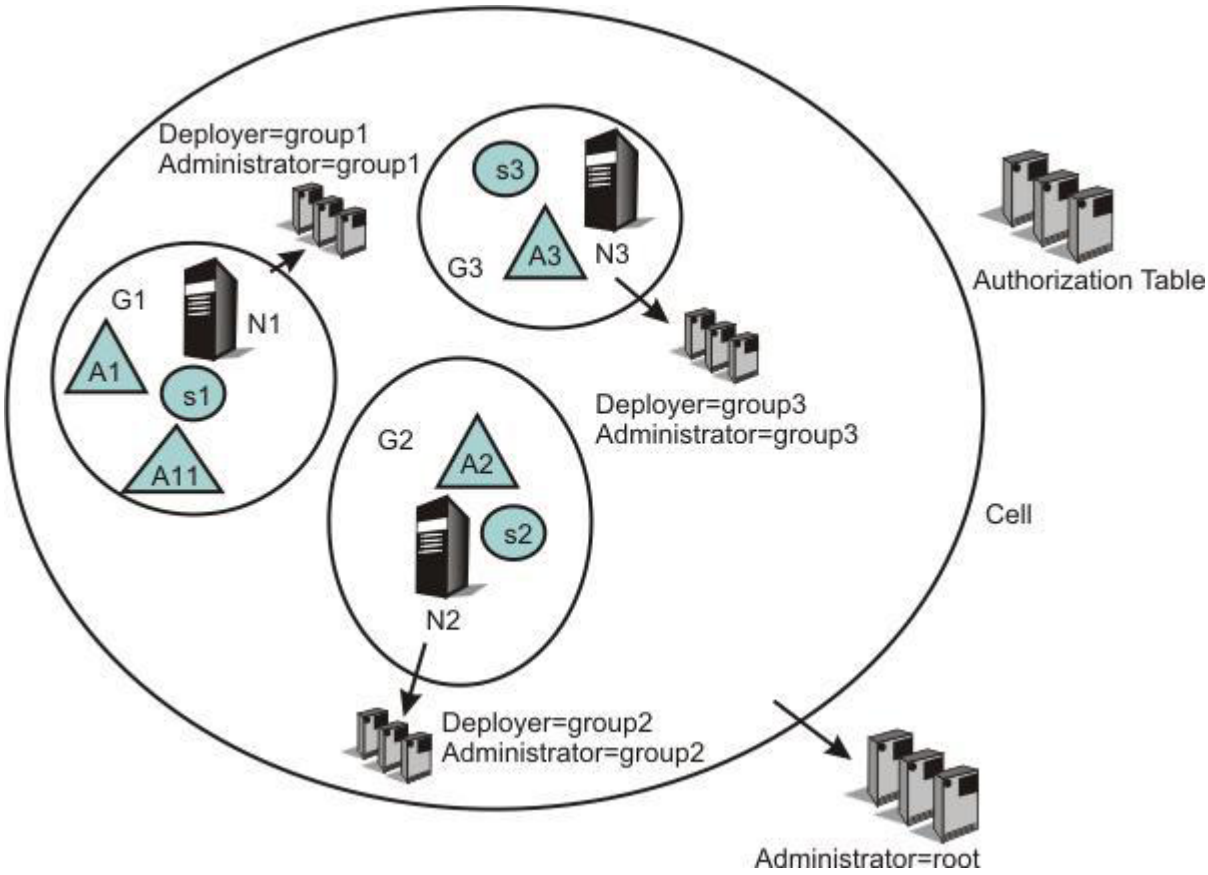
Scenarios that can be applied directly in customer environments.

Each developer must be able to modify the configuration for their server, and they must be able to install their application onto that server. They also must be able to start and stop the server as well as the application on the server.

Developers also must be able to configure the server so that they can debug any problems they run into. They must have the ability to update or modify the application being developed. The administrative authorization group for this developer includes at least one server and any applications that the developer installs on that server.



In the following example, developers of authorization group G1 have a new application (A11). They can install and target that new application only on servers within authorization group G1. Also, they can place that new application in their authorization group (G1).



ASP environment scenario.

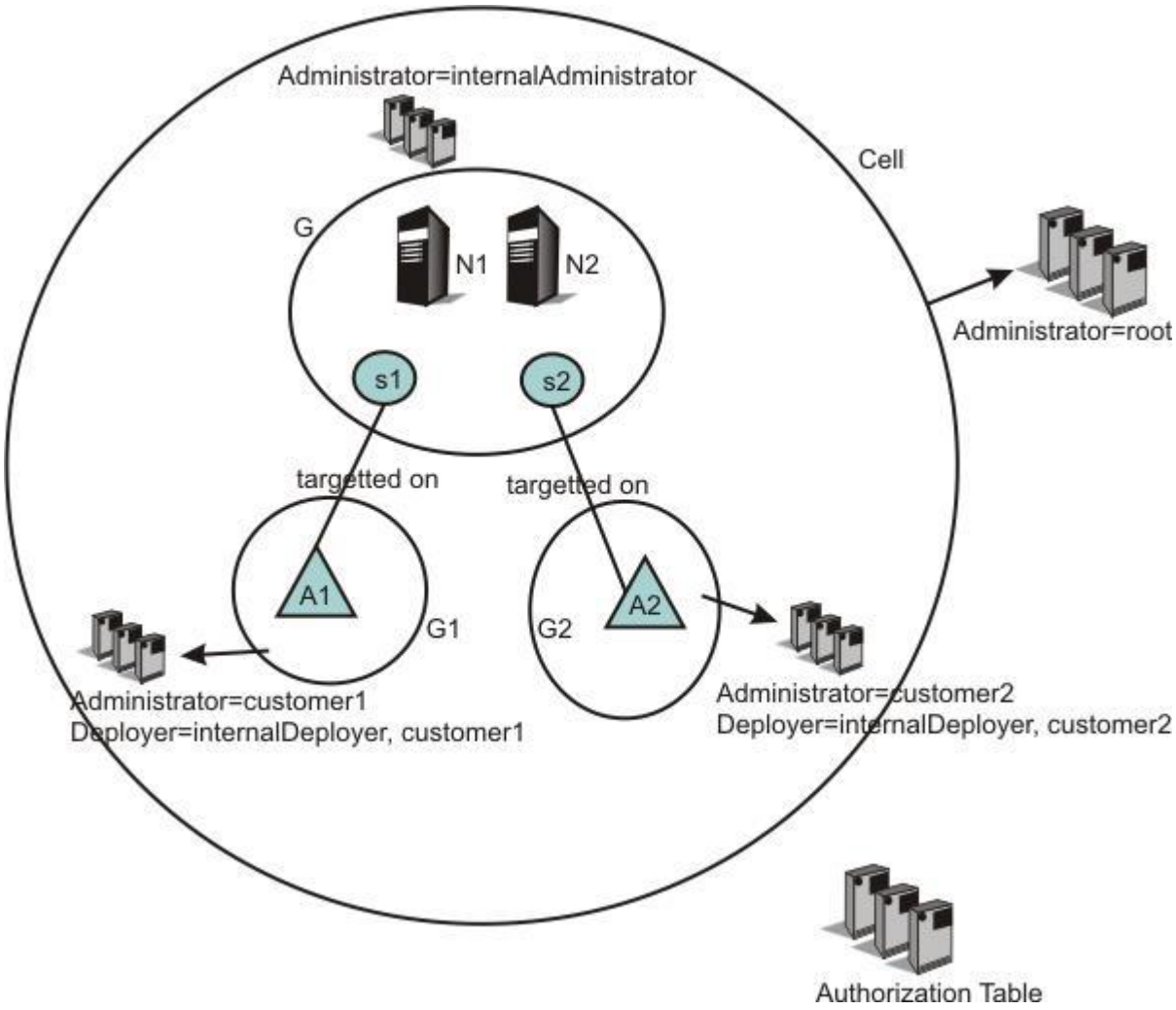
In this scenario, the customer is an ASP. They have their own customers to whom they provide application serving function. They want to enable their customers to administer and monitor their applications, but not to see or administer applications for different customers. In this example, however, the ASP has internal staff administrators whose job it is to maintain the servers.

This internal ASP staff administrator might need to move an application from one server to another to ensure that an application remains available. The internal ASP staff administrator should be able to stop and start the servers and to change their configuration.

In contrast, the ASP customer administrator should not be able to stop or start servers. However, the ASP customer administrator should be able to update their applications running on those servers. The administrative authorization group for the internal ASP administrator can be the whole cell or can include a subset of servers, nodes, clusters and applications. The administrative authorization group for the customer administrator only includes those applications that the customer has paid to have served by this ASP.

When updating the configuration repository, run the admin scripts from the deployment manager so that the fine grain admin security rules will be in effect when admin scripts are run from the deployment manager side.

The following diagram contains a scenario where two different customers have two different types of applications, and can manage their own applications. However, the servers and nodes on which the applications are running are isolated from their customers. The servers and nodes can only be maintained by the internal administrators. In addition, the customers cannot target their applications on a different server. This can only be performed by the internal administrator or internal deployers.

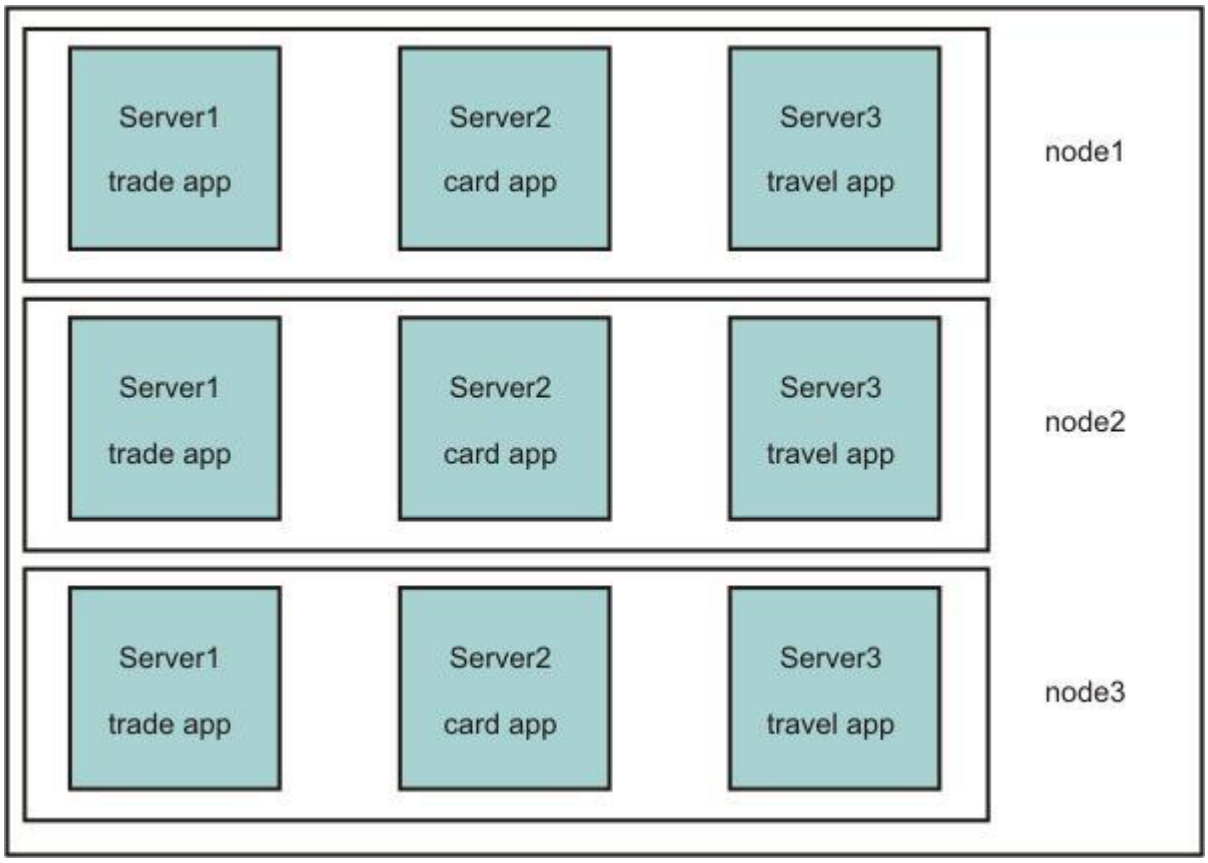


Regional organization scenario.

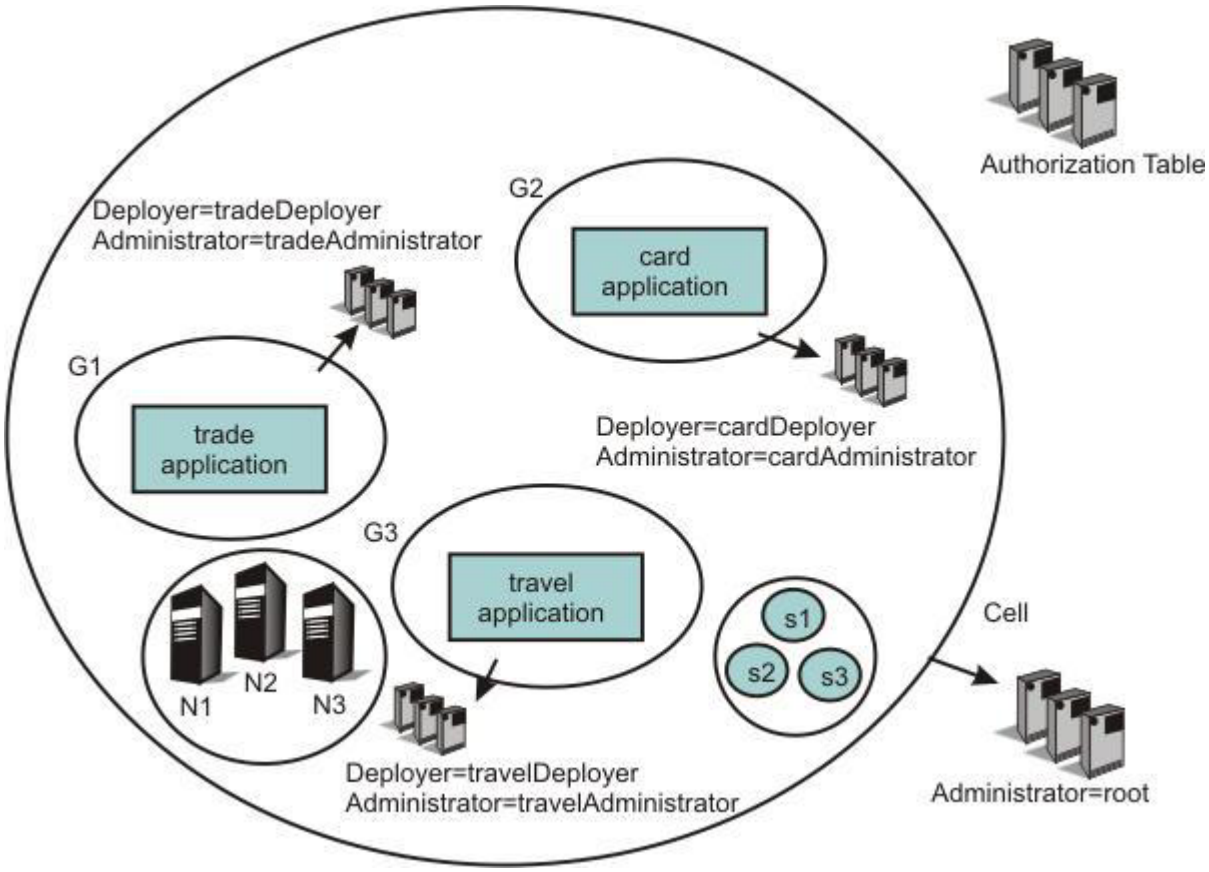
In this scenario, the customer is a large global company. The company's nodes and servers are organized so as to provide application serving for different regions (or alternatively, different lines of business). They want representatives from the different regional areas to be able to monitor and administer the nodes and servers associated with that region. However, they do not want the regional administer to be able to effect any node and server associated with a different region.

The administrative authorization group for each regional representative includes the nodes, servers, clusters and applications associated with that region.

For example, consider a company that provides multiple services, such as a financial institution that provides services like credit card accounts, brokerage accounts, banking accounts, or travel accounts. Each of these services can be separate applications, and the administrator for each of these applications must also be different. The following figure shows one way to configure such a system:

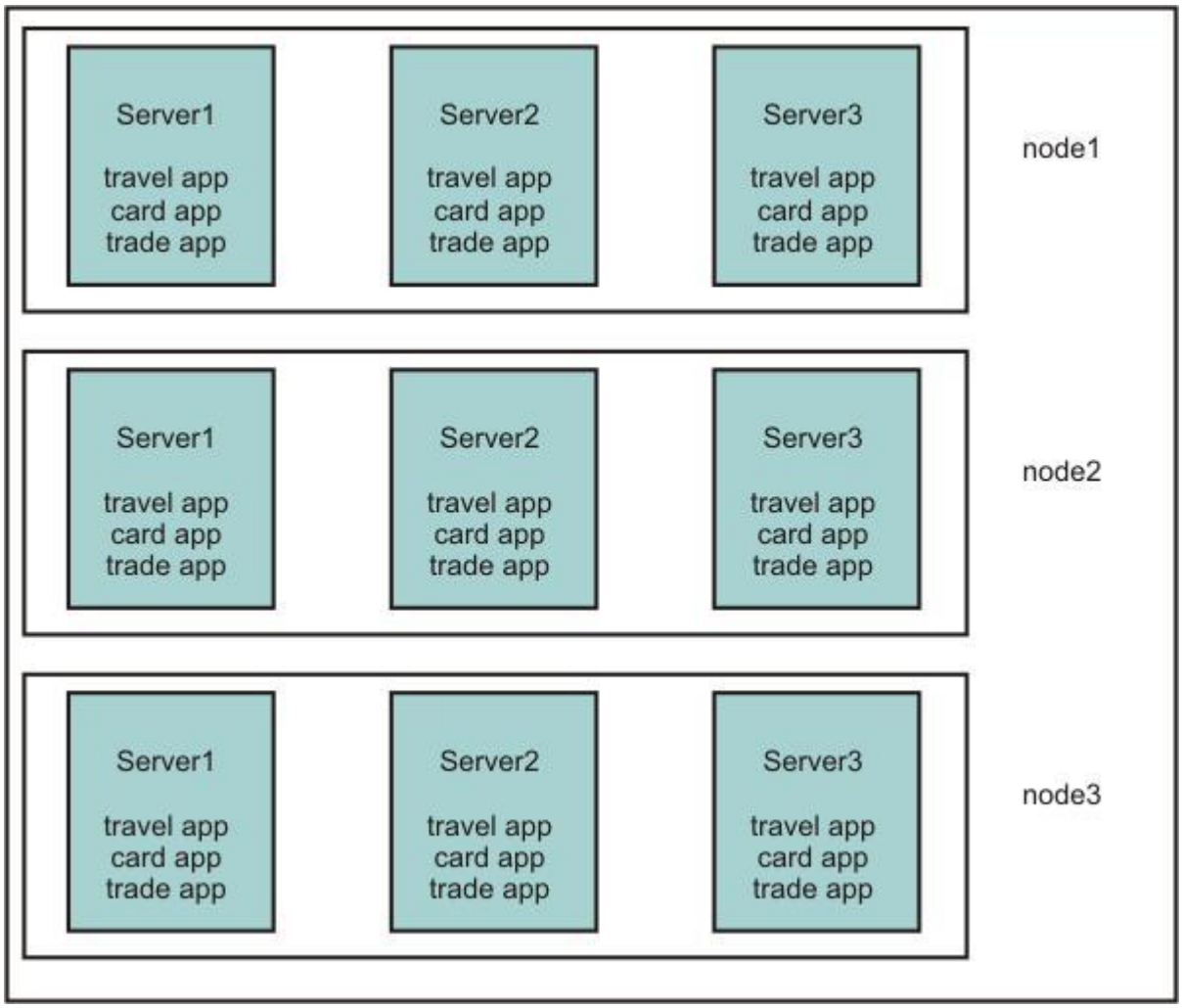


The following figure shows how the resources in such a system can be grouped to isolate administrators from each other:

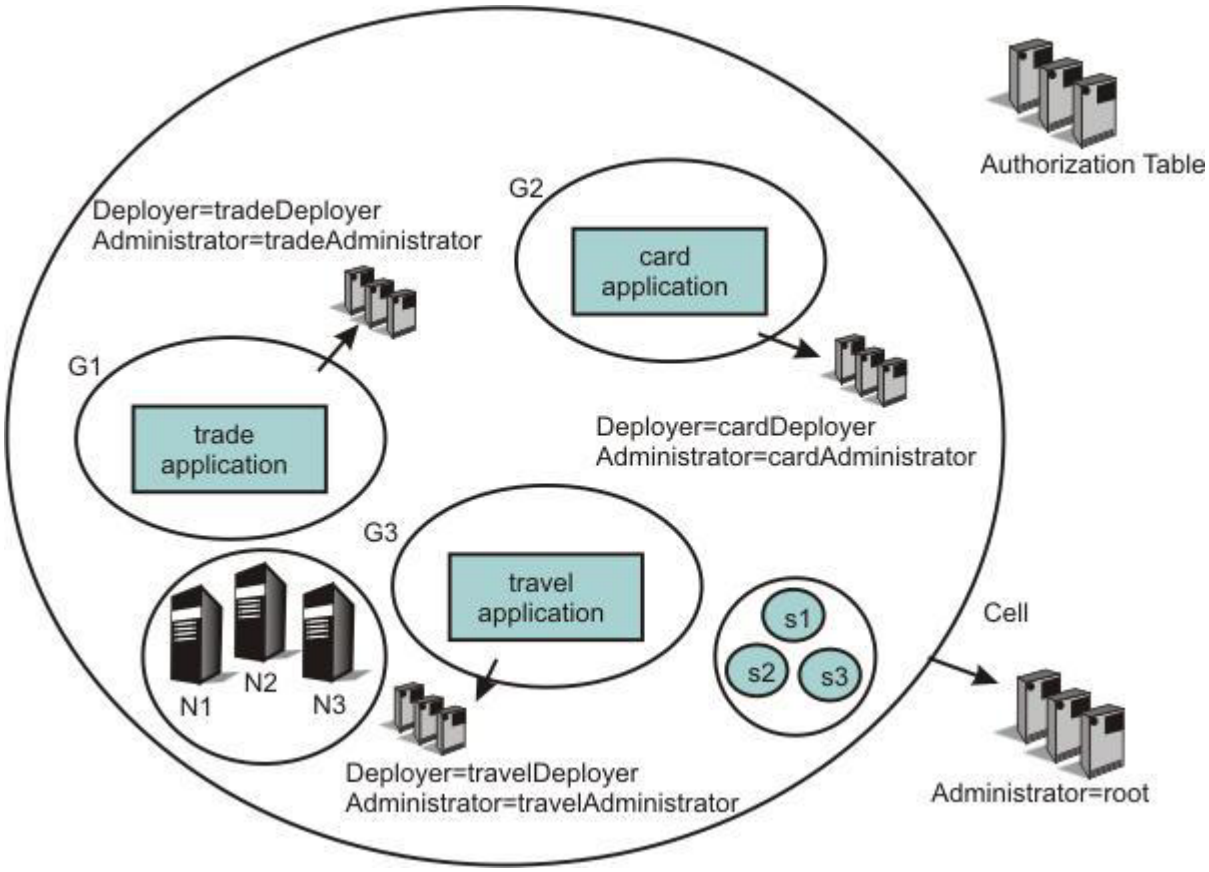


Note that the nodes are not part of any authorization group. Therefore, a trade application administrator cannot stop a server on any of the nodes, and is prevented from stopping a travel application.

The same system can be configured in another way as follows:



The following figure shows how the resources in such a system can be grouped to isolate administrators from each other:



New Administrative Authorization Group

Use this page to create a new administrative authorization group and to specify the associated administrative resources.

To view this administrative console page, click **Security** > **Administrative Authorization Groups** > **New**.

You must be logged into the administrative console with cell-level **AdminSecurityManager** authority, or the primary administrative ID can make these changes as well.

Name

Use to identify the new administrative authorization group. The name should be descriptive of the group's role, or purpose, and should be unique in the cell structure. Using a non-unique name results in an error and a failure to create the new administrative authorization group. This is a required field.

Resources

Select the resources from the Resource section to which you want the new administrative authorization group to control access.

Resources that are displayed in black text are available for selection.

Resources that are displayed in grey are already members of a different administrative authorization group. Therefore, these resources are not available for inclusion in the new administrative authorization group. When a resource is a member of a different authorization group, the name of the group displays next to the resource name. For example: server_1 (group_1)

The available filtering options are the following. Each option includes all the resources that are associated with that specific filtering option.

All scopes
The default view that displays the authorization group tree.

Clusters
All of the resources associated with the clusters.

Web Servers
All of the resources associated with the web servers.

Business-level applications
All of the resources associated with the business-level applications.

Servers
All of the resources associated with the servers.

Nodes
All of the resources associated with the nodes.

Applications
All of the resources associated with the applications.

Assets
All of the resources associated with the assets.

Node Groups
All of the resources associated with the node groups.

Assigned scopes
Displays all of the scopes explicitly assigned to the current authorization group.

Administrative Authorization Group collection

Use this page to create, delete or to edit an existing administrative authorization group.

To view this administrative console page, click **Security** > **Administrative Authorization Groups**.

You must be logged into the administrative console with cell-level **AdminSecurityManager** authority, or the primary administrative ID can make these changes as well.

Name

The name field specifies the current name of the administrative authorization group. You can edit the name of the administrative authorization group during the creation process only. After the authorization group is created, you cannot modify the name. The specified name must be unique within the cell structure. Otherwise, a non-unique name results in an error.

New

Select to create a new administrative authorization group.

Delete

Select to remove an existing administrative authorization group.

Note: You must select an administrative authorization group before selecting **Delete**.

Creating a fine-grained administrative authorization group using the administrative console

You can create a fine-grained administrative authorization group by selecting administrative resources to be part of the authorization group. You can assign users or groups to this new administrative authorization group and also give them access to the administrative resources contained within.

Before you begin

You must be logged into the administrative console with cell-level **Admin Security Manager** authority, or the primary administrative ID can make these changes as well.

Procedure

1. Navigate to **Security** > **Administrative Authorization Groups** > **New**.
2. Type a name for the administrative authorization group into the **Name** field. This is a required field. The name must be unique within the cell structure. If the name is not unique then the new administrative authorization group is not created at the end of this procedure.
3. Select the resources from the Resource section to which you want the new administrative authorization group to control access.

   Resources that are displayed in black text are available for selection.

   Resources that are displayed in grey are already members of a different administrative authorization group. Therefore, these resources are not available for inclusion in the new administrative authorization group. When a resource is a member of a different authorization group, the name of the group displays next to the resource name. For example: server_1 (group_1)

   Your filtering options include the following:

   - Nodes. (All of the resources associated with the nodes.)
   - Servers. (All of the resources associated with the servers.)
   - Web servers. (All of the resources associated with the web servers.)
   - Clusters. (All of the resources associated with the clusters.)
   - Applications. (All of the resources associated with the applications.)
   - Node groups. (All of the resources associated with the Node Groups.)
   - All scopes. (The default view that displays the authorization group tree.)
   - Assigned scopes. (Displays all of the scopes explicitly assigned to the current authorization group.)
4. Click **OK** or **Apply**.
5. If you want to associate a user role to this new administrative authorization group, do the following:
   a. Click **Administrative user roles** located under the Additional Properties section. The available user roles are the following:

      Administrator

      An individual or group that uses the administrator role has the operator and configurator privileges plus additional privileges that are granted solely to the administrator role. For example, an administrator can complete the following tasks:

      - Modify the server user ID and password.
      - Configure authentication and authorization mechanisms.
      - Enable or disable administrative security.
      - Enable or disable Java 2 security.
      - Change the Lightweight Third Party Authentication (LTPA) password and generate keys.
      - Create, update, or delete users in the federated repositories configuration.
      - Create, update, or delete groups in the federated repositories configuration.

      Note: An administrator cannot map users and groups to the administrator roles.
      .

      Configurator

      An individual or group that uses the configurator role has the monitor privilege plus the ability to change the WebSphere Application Server configuration. The configurator can perform all the day-to-day configuration tasks. For example, a configurator can complete the following tasks:

- Create a resource.
- Map an application server.
- Install and uninstall an application.
- Deploy an application.
- Assign users and groups-to-role mapping for applications.
- Set up Java 2 security permissions for applications.
- Customize the Common Secure Interoperability Version 2 (CSIv2), Secure Authentication Service (SAS), and Secure Sockets Layer (SSL) configurations.

  Important: SAS is supported only between Version 6.0.x and previous version servers that have been federated in a Version 6.1 cell.

Deployer
Users granted this role can perform both configuration actions and runtime operations on applications..

Operator
An individual or group that uses the operator role has monitor privileges plus ability to change the runtime state. For example, an operator can complete the following tasks:
- Stop and start the server.
- Monitor the server status in the administrative console.

Monitor
An individual or group that uses the monitor role has the least amount of privileges. A monitor can complete the following tasks:
- View the WebSphere Application Server configuration.
- View the current state of the Application Server.

Admin Security Manager
Using the Admin Security Manager role, you can assign users and groups to the administrative user roles and administrative group roles. However, an administrator cannot assign users and groups to the administrative user roles and administrative group roles including the Admin Security Manager roles.

b. Click **Add...**. The New User page is displayed.
c. Select the appropriate role(s) from the **Role(s)** list box.
d. Select a user or users by entering text in the **Search string** field, and then click **Search.**. Click the arrow to add the available user or users to the **Mapped to role** field. You can select multiple users and roles by clicking **Select All**.
e. Click **OK**. You are returned to the Administrative User Roles page. The new users are displayed in the Administrative User Roles table along with their appropriate roles.
f. Repeat steps B through E for each new user to whom you want to map a role.

6. If you want to associate a group to this new user role, do the following:
   a. Click **Administrative group roles** located under the Additional Properties section.
   b. Click **Add...**. The New Group page is displayed.
   c. Select the appropriate role or roles from the **Role(s)** list box.
   d. Select a user or users by entering text in the **Search string** field, and then click **Search.**. Click the arrow to add the available user or users to the **Mapped to role** field. You can select multiple users and roles by clicking **Select All**.
   e. Select either the **Select from special subjects** or **Map Groups As Specified Below** option.

      If you select the **Select from special subjects** option, you can select the EVERYONE, ALL AUTHENTICATED, or ALL AUTHENTICATED IN TRUSTED REALMS values.

A list of user groups and roles are displayed in the **Available** and **Mapped to role** fields. Select the user groups from the **Available** field and then select the roles from the **Mapped to role** field to which you want the group or groups associated. You can select multiple groups and roles.

f. Click **OK**. You are returned to the Administrative Group Roles page. The new group is displayed in the Administrative Group Roles table along with the role of the new group.

g. Repeat steps B through E for each new group to whom you want to map a role.

7. If you want to create another administrative authorization group, click **Apply**. The current administrative authorization group is created. Repeat steps 2 through 6 to create another administrative authorization group.
8. If you do not want to create another administrative authorization group, click **OK**.

Editing a fine-grained administrative authorization group using the administrative console

You can add or remove administrative resources to an administrative authorization group or edit an existing one.

Before you begin

You must be logged into the administrative console with the cell-level **AdminSecurityManager** authority or as the primary administrative user.

Procedure

1. Navigate to **Security** > **Administrative Authorization Groups**. The Administrative Authorization Groups page displays a table that lists all of the current administrative authorization groups available in the cell.
2. Click on the administrative authorization group in the table that you want to edit.
3. To add or remove resources from the administrative authorization group, select or clear them in the Resource section of the edit page. Resources displayed in black text are available for selection or clearing. Resources displayed in grey text are members of a different administrative authorization group and therefore cannot be edited for the current administrative authorization group.

   The available filtering options are the following. Each option includes all the resources that are associated with that specific filtering option.

   - All scopes. (The default view that displays the authorization group tree.)
   - Clusters. (All of the resources associated with the clusters.)
   - Web servers. (All of the resources associated with the Web servers.)
   - Business-level applications. (All of the resources associated with the business-level applications.)
   - Servers. (All of the resources associated with the servers.)
   - Nodes. (All of the resources associated with the nodes.)
   - Applications. (All of the resources associated with the applications.)
   - Assets. (All of the resources associated with the assets.)
   - Node groups. (All of the resources associated with the node groups.)
   - Assigned scopes. (Displays all of the scopes explicitly assigned to the current authorization group).

   Nodes prior to WebSphere Application Server Version 6.1 in a mixed cell environment are filtered out of resource mapping.
4. To remove a user or a group, do the following:

   a. To delete users, click **Administrative user roles** under the Additional Properties section. To delete groups, click **Administrative group roles** under the Additional Properties section. The appropriate edit page displays a table that lists all of the current users or groups and their associated roles, along with the user's login status.

b. Click the check box beside the name of the current user or group and then click **Remove**. The current user or group is no longer associated with the role and the role is no longer listed in the table. It is now ready to have a new user or group assigned to it.

5. If you want to add or to reassign a user or group role to this administrative authorization group, do the following:
   a. To add a user, click **Administrative user roles** under the Additional Properties section. To add a group, click **Administrative group roles** located under the Additional Properties section. The appropriate edit page displays a table that lists all of the current users or groups and their associated roles. The available roles are:

      Administrator
      An individual or group that uses the administrator role has the operator and configurator privileges plus additional privileges that are granted solely to the administrator role. For example, an administrator can complete the following tasks:
      - Modify the server user ID and password.
      - Configure authentication and authorization mechanisms.
      - Enable or disable administrative security.
      - Enable or disable Java 2 security.
      - Change the Lightweight Third Party Authentication (LTPA) password and generate keys.
      - Create, update, or delete users in the federated repositories configuration.
      - Create, update, or delete groups in the federated repositories configuration.

      Note: An administrator cannot map users and groups to the administrator roles.

      Configurator
      An individual or group that uses the configurator role has the monitor privilege plus the ability to change the WebSphere Application Server configuration. The configurator can perform all the day-to-day configuration tasks. For example, a configurator can complete the following tasks:
      - Create a resource.
      - Map an application server.
      - Install and uninstall an application.
      - Deploy an application.
      - Assign users and groups-to-role mapping for applications.
      - Set up Java 2 security permissions for applications.
      - Customize the Common Secure Interoperability Version 2 (CSIv2), Security Authentication Service (SAS), and Secure Sockets Layer (SSL) configurations.

        Important: SAS is supported only between Version 6.0.x and previous version servers that have been federated in a Version 6.1 cell.

      Deployer
      Users granted this role can perform both configuration actions and runtime operations on applications.

      Operator
      An individual or group that uses the operator role has monitor privileges plus ability to change the runtime state. For example, an operator can complete the following tasks:
      - Stop and start the server.
      - Monitor the server status in the administrative console.

      .

Monitor

An individual or group that uses the monitor role has the least amount of privileges. A monitor can complete the following tasks:

- View the WebSphere Application Server configuration.
- View the current state of the Application Server.

Admin Security Manager

Using the Admin Security Manager role, you can assign users and groups to the administrative user roles and administrative group roles. However, an administrator cannot assign users and groups to the administrative user roles and administrative group roles including the Admin Security Manager role.

b. Click **Add...**.

c. To add a new user or group, follow the instructions on the page to specify either a user name, group name, or Special subject. Highlight the desired role(s), and click **OK**. The specified users, groups, or Special subject are mapped to the security roles.

Fine-grained administrative security in heterogeneous and single-server environments

You can use fine-grained administrative security in heterogeneous or single-server environments. This capability enables you to use fine-grained administrative security for nodes that were created on different versions of the product, and applications that are grouped and placed in different authorization groups.

Fine-grained administrative security in a heterogeneous environment

Fine-grained administrative security in a heterogeneous environment has the following requirements:

- Only nodes that are running WebSphere Application Server Version 8.5 can be part of an administrative authorization group.
- Only servers that are running in a WebSphere Application Server Version 8.5 node can be part of an administrative authorization group.
- Only applications that are targeted on servers running on WebSphere Application Server Version 8.5 can be part of an administrative authorization group.
- If a cluster spans nodes of multiple releases, it cannot be part of an administrative authorization group.
- If a cluster spans nodes of multiple releases, none of its members can be part of an administrative authorization group.
- If an application is targeted on a cluster that spans multiple releases, that application cannot be part of an administrative authorization group.

Fine-grained administrative security in a single-server environment

You can also use fine-grained administrative security in a single-server environment. This capability means that you can group various applications in the single server, and place them in different authorization groups. Therefore, different authorization constraints might exist for different applications.

Life cycle of fine-grained administrative resource

An administrative resource that was once part of an authorization group continues to be part of that authorization group until one of the following events occurs:

- The administrative resource is removed from the authorization group. In this instance, the administrative resource belongs to the cell-level authorization group.
- The administrative resource is removed from the configuration. In this instance, the administrative resource does not exist in the configuration, but still exists in the authorization group. Remove this administrative resource from the authorization group.

After the administrative resource is removed from the authorization group, the administrative authorizer runtime must be notified by using the AuthorizationManager refreshAll MBean method.

The **refreshAll** command must be invoked after AdminConfig.save() and sync nodes. For example:

JACL:

```
// get AuthorizationGroup Mbean
wsadmin> set agBean [$AdminControl queryNames
type=AuthorizationGroupManager,process=dmgr,*]

wsadmin> $AdminControl invoke &agBean refreshAll
```

JYTHON:

```
// get AuthorizationGroup Mbean
wsadmin> set agBean
AdminControl.queryNames('type=AuthorizationGroupManager,process=dmgr,*')

wsadmin> AdminControl.invoke(agBean, 'refreshAll')
```

The server restart is no longer needed.

Using SCA authorization and security identity policies

Use two Service Component Architecture (SCA) declarative policies (*authorization* and *security identity*) to protect SCA components and operations and to declare the security identity under which the SCA components or operations are executed.

Before you begin

A user registry must be configured and an SCA component must first have been developed. You must also enable application security.

About this task

An authorization policy controls who can access protected SCA components and operations. A security identity policy declares the security identity under which an SCA component or operation is executed. You can limit access to an SCA component or to an operation to particular users or groups. You can also delegate access to another user when executing an SCA component or an operation.

Note the following limitations:

- SCA authorization policy is not supported for composites packaged in web application archives (WAR files).
- The `definitions.xml` file must be packaged in the same asset as the composites that reference its policy sets.

  For OASIS SCA applications, `definitions.xml` must be packaged in the same asset as the composites that reference its policy sets under the `META-INF` directory.
- Role assignments are scoped to a configuration unit, and are required for all of the roles used in all of the composites within the configuration unit. These role assignments are completely independent of any role assignments made for other configuration units in the same business-level application.
- The target namespace of the policy set and the name of the policy set do not contribute to the name of a role. They are used solely to resolve the policy set reference. This implies that within the same configuration unit, identically-named roles that are defined within different policy sets or different name spaces are treated as the same role.
- If authorization policy is not attached to a given component and operation, the operation runs unprotected.
- It is possible to create conflicts by specifying multiple policy sets in the @policySets attribute or by inheriting policy sets across elements. In this case, the following rules are used:

  - The **<denyAll>** element takes precedence over **<permitAll>**, which takes precedence over **<allow>**.
  - Roles from multiple **<allow>** elements are aggregated.
- SCA authorization policy does not support authorizing users in foreign realms.
- The OASIS SCA specification permits you to specify the authorization intent on the implementation. If the implementation contains an authorization intent, you must attach a policy set that satisfies the intent to the implementation. The attached policy set must contain the `provides="authorization"` statement in its definition.
- In OASIS SCA, the security elements that a policy set can contain, such as `authorization`, `securityIdentity`, `allow`, and `runAs`, belong to the Tuscany namespace `http://tuscany.apache.org/xmlns/sca/1.1`.

Access to an SCA component is permitted or denied by using the following steps:

Procedure

1. The policy administrator creates one or more policy sets in the file named `definitions.xml`.

   OSOA example

```
<definitions xmlns="http://www.osoa.org/xmlns/sca/1.0"
  targetNamespace="http://smallvilleBank"
  xmlns:sca="http://www.osoa.org/xmlns/sca/1.0">
  <policySet name="StaffAuthorizationPolicy"
  appliesTo="sca:implementation.java"
  xmlns="http://www.osoa.org/xmlns/sca/1.0">
    <authorization>
      <allow roles="staff"/>
    </authorization>
  </policySet>
  <policySet name="SupervisorAuthorizationPolicy"
  appliesTo="sca:implementation.java"
  xmlns="http://www.osoa.org/xmlns/sca/1.0">
    <authorization>
      <allow roles="supervisor manager specialist"/>
    </authorization>
    <securityIdentity>
      <runAs role="specialist"/>
    </securityIdentity>
 </policySet>
</definitions>
```

   OASIS example

```
<definitions xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912"
  targetNamespace="http://smallvilleBank"
  xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200912"
  xmlns:tuscany="http://tuscany.apache.org/xmlns/sca/1.1">
  <policySet name="StaffAuthorizationPolicy"
       appliesTo="sca:implementation.java"
       provides="authorization">
    <tuscany:authorization>
       <tuscany:allow roles="staff"/>
    </tuscany:authorization>
  </policySet>
  <policySet name="SupervisorAuthorizationPolicy"
       appliesTo="sca:implementation.java">
    <tuscany:authorization>
       <tuscany:allow roles="supervisor manager specialist"/>
    </tuscany:authorization>
    <tuscany:securityIdentity>
       <tuscany:runAs role="specialist"/>
    </tuscany:securityIdentity>
  </policySet>
</definitions>
```

2. The assembler attaches the policy to the SCA composite.

   OSOA example

```
<?xml version="1.0" encoding="UTF-8"?>
<composite xmlns="http://www.osoa.org/xmlns/sca/1.0"
  xmlns:bank="http://smallvilleBank"
```

```
  name="AccountServices">
  <component name="AccountAccess">
        <implementation.java class="smallvilleBank.AccountAccessImpl"
             policySets="bank:StaffAuthorizationPolicy"/>
  </component>
  <component name="AccountAudit">
        <implementation.java class="smallvilleBank.AccountAuditImpl"
             policySets="bank:SupervisorAuthorizationPolicy"/>
  </component>
</composite>
```

OASIS example

```
<?xml version="1.0" encoding="UTF-8"?>
<composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912"
  xmlns:bank="http://smallvilleBank"
  name="AccountServices">
  <component name="AccountAccess">
    <implementation.java class="smallvilleBank.AccountAccessImpl"
       requires="authorization" policySets="bank:StaffAuthorizationPolicy"/>
  </component>
  <component name="AccountAudit">
    <implementation.java class="smallvilleBank.AccountAuditImpl"
       policySets="bank:SupervisorAuthorizationPolicy"/>
  </component>
</composite>
```

3. The deployer assigns users and or groups to the roles that are defined in the composite.
4. The deployer assigns a user to the `runAs` roles that are defined in the composite.

What to do next

Access to the SCA component is permitted or denied according to the authorization policy.

Using the SCA RequestContext.getSecuritySubject() API

The Service Component Architecture (SCA) RequestContext.getSecuritySubject() application programming interface returns a Java Authentication and Authorization (JAAS) subject that represents an authenticated user who accesses the protected SCA service.

Before you begin

SCA service developers can use the RequestContext.getSecuritySubject() API to obtain a JAAS Subject that represents the requester.

If one or more of the following preconditions are not met the SCA request is not authenticated, and the RequestContext.getSecuritySubject API returns a null Subject:

- Administrative security must be enabled to initialize the security infrastructure.
- Application security must be enabled to enforce security policy and authentication.
- The SCA service must require an authenticated user. Authentication can be done at the transport layer using the `authentication.transport` intent (for OSOA composites) or the `clientAuthentication.transport` intent (for OASIS composites). Authentication can be done at the message layer by attaching a web service policy set that requires authentication.

About this task

When using the RequestContext.getSecuritySubject() API, perform the following steps:

Procedure

1. Use the RequestContext.getSecuritySubject API in your file.

   The following example utilizes the OSOA RequestContext.getSecuritySubject API:

```
import org.osoa.sca.annotations.Context;
import org.osoa.sca.annotations.Service;
import org.osoa.sca.RequestContext;
import javax.security.auth.Subject;
import java.security.Principal;
import java.util.Iterator;
import com.ibm.websphere.security.cred.WSCredential;

@Service(EchoService.class)
public class EchoServiceWithIdentityComponentImpl implements EchoService
{
    @Context
    protected RequestContext requestContext;

    public String echo_String(String input)
    {
        try {
            Subject subject = null;
            String securityName = null;

            if (requestContext != null) {
                subject = requestContext.getSecuritySubject();
             }

            if (subject != null) {
                 java.util.Set principalSet = subject.getPrincipals();
                 if (principalSet != null && principalSet.size() > 0) {
                     Iterator principalIterator = principalSet.iterator();
                     if (principalIterator.hasNext()) {
                         Principal principal = (java.security.Principal) principalIterator.next();
                         securityName = principal.getName();
                     }
                 }
            }
         } catch (Exception ex) {
              // Handle exception
         }
     }
}
```

The same example applies to using the OASIS RequestContext.getSecuritySubject API with the exception of package name changes:

```
import org.oasisopen.sca.annotation.Context;
import org.oasisopen.sca.annotation.Service;
import org.oasisopen.sca.RequestContext;
```

2. You can obtain various security attributes of the request from the WSCredential object in the subject as shown in the following example:

```
if (subject != null) {
    java.util.Set credSet = subject.getPublicCredentials();
    if (credSet != null && credSet.size() > 0)
    {
        Iterator credIterator = credSet.iterator();
        while (credIterator.hasNext()) {
            Object o = credIterator.next();
            WSCredential cred = null;
            if (o instanceof WSCredential) {
                cred = (WSCredential) o;
            } else {
                if (securityName == null) {
                    securityName = new StringBuffer();
                }
                securityName.append("\n>> Found a public credential: " + o.getClass().getName());
            }
            if (cred != null) {
                if (securityName == null) {
                    securityName = new StringBuffer();
                }
                securityName.append("\n>> WSCredential security attributes . . .");
                securityName.append("\n>> getAccessId = \t\t" + cred.getAccessId());
                securityName.append("\n>> getGroupIds = \t\t" + cred.getGroupIds());
                securityName.append("\n>> getPrimaryGroupId = \t\t" + cred.getPrimaryGroupId());
                securityName.append("\n>> getRealmName = \t\t" + cred.getRealmName());
```

```
                securityName.append("\n>> getRealmSecurityName = \t\t" + cred.getRealmSecurityName());
                securityName.append("\n>> getRealmUniqueSecurityName = \t\t" + cred.getRealmUniqueSecurityName());
                securityName.append("\n>> getSecurityName = \t\t" + cred.getSecurityName());
                securityName.append("\n>> getUniqueSecurityName = \t\t" + cred.getUniqueSecurityName());
            }
        }
    }
}
```

The principal identity consists of a realm name followed by the identity of the requester. For example, assume WebSphere Application Server is configured to use an Lightweight Directory Access Protocol (LDAP) server for authentication. The realm name is the LDAP server host name and the port number:

```
security name = ldap1.austin.ibm.com:389/user2
```

Sample output is shown below:

```
>> WSCredential security attributes . . .
>> getAccessId =    user:ldap1.austin.ibm.com:389/cn=user2,o=ibm,c=us
>> getGroupIds =    [group:ldap1.austin.ibm.com:389/CN=GROUP2,O=IBM,C=US]
>> getPrimaryGroupId =   group:ldap1.austin.ibm.com:389/CN=GROUP2,O=IBM,C=US
>> getRealmName =   ldap1.austin.ibm.com:389
>> getRealmSecurityName =  ldap1.austin.ibm.com:389/user2
>> getRealmUniqueSecurityName = ldap1.austin.ibm.com:389/cn=user2,o=ibm,c=us
>> getSecurityName =   user2
>> getUniqueSecurityName =  cn=user2,o=ibm,c=us
```

Chapter 9. Securing communications

WebSphere Application Server provides several methods to secure communication between a server and a client.

About this task

Note: WebSphere Application Server provides several methods for securing communication between a server and a client. New in this release are functions that ensure secure communication between a server and a client. These functions focus on certificate management, authentication, and ensuring trust among the application server, administrative agent, and job manager. The new functions include:

- Creating and using a certificate authority (CA) clients to enable a CA to request, query, and revoke certificates.
- Creating and using chained personal certificates to allow a certificate to be signed with a longer life span.
- Creating and revoking certificate authority (CA) certificates to ensure secure communication between the CA client and the CA server.

The following topics are covered in this section:

Procedure

- Secure communications using Secure Sockets Layer
- Creating an SSL configuration
- Creating a keystore configuration
- Creating a certificate authority (CA) client
- Deleting a certificate authority (CA) client
- Viewing or Modifying a certificate authority (CA) client
- Creating a keystore configuration for a preexisting keystore file
- Creating a self-signed certificate
- Creating a certificate authority request
- Extracting a signer certificate from a personal certificate
- Retrieving signers from a remote SSL port
- Adding a signer certificate to a keystore
- Adding a signer certificate to the default signers keystore
- Exchanging signer certificates in a keystore
- Configuring certificate expiration monitoring
- Key management for cryptographic uses
- Creating a key set configuration
- Creating a key set group configuration
- Configuring the web server plug-in for Secure Sockets Layer

Secure communications using Secure Sockets Layer (SSL)

The Secure Sockets Layer (SSL) protocol provides transport layer security including authenticity, data signing, and data encryption to ensure a secure connection between a client and server that uses WebSphere Application Server. The foundation technology for SSL is public key cryptography, which guarantees that when an entity encrypts data using its public key, only entities with the corresponding private key can decrypt that data.

WebSphere Application Server uses Java Secure Sockets Extension (JSSE) as the SSL implementation for secure connections. JSSE is part of the Java 2 Standard Edition (J2SE) specification and is included in the IBM implementation of the Java Runtime Extension (JRE). JSSE handles the handshake negotiation and protection capabilities that are provided by SSL to ensure secure connectivity exists across most protocols. JSSE relies on X.509 certificate-based asymmetric key pairs for secure connection protection and some data encryption. Key pairs effectively encrypt session-based secret keys that encrypt larger blocks of data. The SSL implementation manages the X.509 certificates.

Managing X.509 certificates

Secure communications for WebSphere Application Server require digitally-signed X.509 certificates. The contents of an X.509 certificate, such as its distinguished name and expiration, are either signed by a certificate authority (CA), signed by a root certificate in **NodeDefaultRootStore** or **DmgrDefaultRootStore**, or are self-signed. When a trusted CA signs an X.509 certificate, WebSphere Application Server identifies the certificate and freely distributes it. A certificate must be signed by a CA because the certificate represents the identity of an entity to the general public. Server-side ports that accept connections from the general public must use CA-signed certificates. Most clients or browsers already have the signer certificate that can validate the X.509 certificate so signer exchange is not necessary for a successful connection.

You can trust the identity of a self-signed X.509 certificate only within a peer in a controlled environment, such as internal network communications, accepts the signer certificate. To complete a trusted handshake, you must first send a copy of the entity certificate to every peer that connects to the entity.

CA, chained, and self-signed X.509 certificates reside in Java **keystores**. JSSE provides a reference to the keystore in which a certificate resides. You can select from many types of keystores, including Java Cryptographic Extension (JCE)-based and hardware-based keystores. Typically, each JSSE configuration has two Java keystore references: a keystore and a **truststore**. The keystore reference represents a Java keystore object that holds personal certificates. The truststore reference represents a Java keystore object that holds signer certificates.

A personal certificate without a private key is an X.509 certificate that represents the entity that owns it during a handshake. Personal certificates contain both public and private keys. A signer certificate is an X.509 certificate that represents a peer entity or itself. Signer certificates contain just the public key and verify the signature of the identity that is received during a peer-to-peer handshake.

For more information, see “Extracting a signer certificate from a personal certificate” on page 790

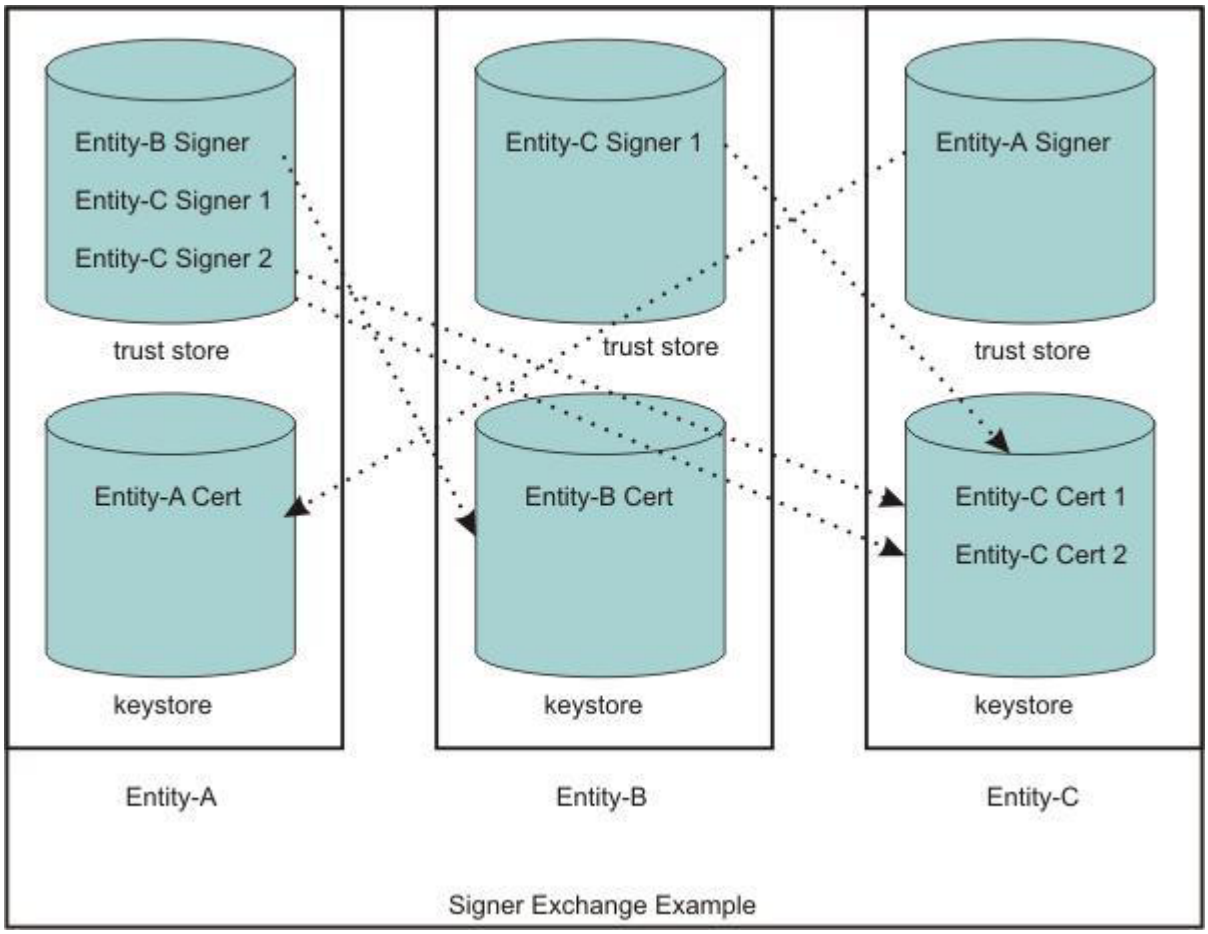
For more information about keystores, see Keystore configurations for SSL.

Signer exchange

When you configure an SSL connection, you can exchange signers to establish trust in a **personal certificate** for a specific entity. Signer exchange enables you to extract the X.509 certificate from the peer keystore and add it into the truststore of another entity so that the two peer entities can connect. The signer certificate also can originate from a CA as a root signer certificate or a chained certificate's root signer certificate or an intermediate signer certificate. You can also extract a **signer certificate** directly from a self-signed certificate, which is the X.509 certificate with the public key.

Figure 1 illustrates a hypothetical keystore and truststore configuration. An SSL configuration determines which entities can connect to other entities, and the peer connections that are trusted by an SSL handshake. If you do not have the necessary signer certificate, the handshake fails because the peer cannot be trusted.

Figure 35. Signer exchange



In this example, the truststore for Entity A contains three signers. Entity A can connect to any peer as long as one of the three signers validates its personal certificate. For example, Entity A can connect to Entity B or Entity C because the signers can trust both signed personal certificates. The truststore for Entity-B contains one signer. Entity B is able to connect to Entity C only, and only when the peer endpoint is using certificate Entity-C Cert 1 as its identity. The ports that use the other personal certificate for Entity C are not trusted by Entity B. Entity C can connect to Entity A only.

In the example, the self-signed configuration seems to represent a one-to-one relationship between the signer and the certificate. However, when a CA signs a certificate, it typically signs many at a time. The advantage of using a single CA signer is that it can validate personal certificates that are generated by the CA for use by peers. However, if the signer is a public CA, you must be aware that the signed certificates might have been generated for another company other than your target entity. For your internal communications, private CAs and self-signed certificates are preferable to public CAs because they enable you to isolate the connections that you want to occur and prevent those that you do not want to occur.

SSL configurations

An SSL configuration comprises a set of configuration attributes that you can associate with an endpoint or set of endpoints in the WebSphere Application Server topology. The SSL configuration enables you to create an SSLContext object, which is the fundamental JSSE object that the server uses to obtain SSL socket factories. You can manage the following configuration attributes:

- An alias for the SSLContext object
- A handshake protocol version
- A keystore reference
- A truststore reference

- A key manager
- One or more trust managers
- A security level selection of a cipher suite grouping or a specific cipher suite list
- A certificate alias choice for client and server connections

To understand the specifics of each SSL configuration attribute, see “SSL configurations” on page 674.

Selecting SSL configurations

In previous releases of WebSphere Application Server, you can reference an SSL configuration only by selecting the SSL configuration alias directly. Each secure endpoint was denoted by an alias attribute that references a valid SSL configuration within a repertoire of SSL configurations. When you made a single configuration change, you had to re-configure many alias references across the various processes. Although the current release still supports direct selection, this approach is no longer recommended.

The current release provides improved capabilities for managing SSL configurations and more flexibility when you select SSL configurations. In this release, you can select from the following approaches:

Programmatic selection

You can set an SSL configuration on the running thread prior to an outbound connection. WebSphere Application Server ensures that most system protocols, including Internet Inter-ORB Protocol (IIOP), Java Message Service (JMS), Hyper Text Transfer Protocol (HTTP), and Lightweight Directory Access Protocol (LDAP), accept the configuration. See “Programmatically specifying an outbound SSL configuration using JSSEHelper API” on page 744

Dynamic selection

You can associate an SSL configuration dynamically with a specific target host, port, or outbound protocol by using a predefined selection criteria. When it establishes the connection, WebSphere Application Server checks to see if the target host and port match a predefined criteria that includes the domain portion of the host. Additionally, you can predefine the protocol for a specific outbound SSL configuration and certificate alias selection. See “Dynamic outbound selection of Secure Sockets Layer configurations” on page 685 for more information.

Dynamic outbound selection of Secure Sockets Layer configurations is based on connection information being available for the server so that the server can match up the outbound protocol, host, or port when the creation of the client socket takes place in com.ibm.websphere.ssl.protocol.SSLSocketFactory. For WebSphere admin connectors like SOAP and Remote Method Invocation (RMI), connection information is placed on the thread and is available for dynamic outbound selection to take place. The dynamic outbound selection process replies on connection information being setup when SSL properties are retrieved similar to what is described in “Programmatically specifying an outbound SSL configuration using JSSEHelper API” on page 744.

When the outbound connection is being made from customer written applications, parts of the connection information may not be available. Some of these applications make API calls to a protocol to make the connection. The API ultimately then calls one of the createSocket() methods in com.ibm.websphere.ssl.protocol.SSLSocketFactory to complete the process. gotcha: All of the connection information for dynamic outbound selection might not be available, and you may have to adjust the dynamic outbound selection connection filter and fill in an asterisk (*) for the missing part of the connection information. As an example, the openConnection() call on a URL object ultimately calls `createSocket(java.net.Socket socket, String host, int port, boolean autoClose)`. The connection information can be built with the host and port provided, but there is no protocol provided. In this case, a wild card, asterisk (*), should be used for the protocol part of the dynamic selection connection information.

Most of the createSocket() methods take a host or IP address and a port as parameters. The dynamic outbound selection connection filter can be built with the host and port. The default method, createSocket(), without any parameters does not contain any information to build the

outbound selection connection filter unless the socket factory was instantiated with connection information, If no connection information is available, then you should consider using one of the other methods of selecting a SSL configuration describes in this topic, "Programmatic selection" can be good choice.

Direct selection

You can select an SSL configuration by using a specific alias, as in past releases. This method of selection is maintained for backwards compatibility because many applications and processes rely on alias references.

Scope selection

You can associate an SSL configuration and its certificate alias, which is located in the keystore associated with that SSL configuration, with a WebSphere Application Server management scope. This approach is recommended to manage SSL configurations centrally. You can manage endpoints more efficiently because they are located in one topology view of the cell. The inheritance relationship between scopes reduces the number of SSL configuration assignments that you must set.

Each time you associate an SSL configuration with a cell scope, the node scope within the cell automatically inherits the configuration properties. However, when you assign an SSL configuration to a node, the node configuration overrides the configuration that the node inherits from the cell. Similarly, all of the application servers for a node automatically inherit the SSL configuration for that node unless you override these assignments. Unless you override a specific configuration, the topology relies on the rules of inheritance from the cell level down to the endpoint level for each application server.

Note: If your applications are relying on SSL configurations that were set as individual settings for each SSL configuration in the topology, but your application servers have inherited the SSL configuration as deployed from the cell level down to the endpoint level, then there is the possibility of communication errors occurring among servers (for example, handshake errors). You need to ensure that your applications are operating consistent with the central management of SSL configurations.

The topology view displays an inbound tree and outbound tree. You can make different SSL configuration selections for each side of the SSL connection based on what that server connects to as an outbound connection and what the server connects to as an inbound connection. See “Central management of SSL configurations” on page 686 for more information.

The runtime uses an order of precedence for determining which SSL configuration to choose because you have many ways to select SSL configurations. Consider the following order of precedence when you select a configuration approach:

1. Programmatic selection. If an application sets an SSL configuration on the running thread using the com.ibm.websphere.ssl.JSSEHelper application programming interface (API), the SSL configuration is guaranteed the highest precedence.
2. Dynamic selection criteria for outbound host and port or protocol.
3. Direct selection.
4. Scope selection. Scope inheritance guarantees that the endpoint that you select is associated with an SSL configuration and is inherited by every scope beneath it that does not override this selection.

Default chained certificate configuration

By default, WebSphere Application Server creates a unique chained certificate for each node. The chained certificate is signed with a root, a self-signed certificate stored in the **DmgrDefaultRootStore** or **NodeDefaultRootStore**. WebSphere Application Server no longer relies on a self-signed certificate or the default or dummy certificate that is shipped with the product. The `key.p12` default keystore and the trust.p12 truststore are stored in the configuration repository within the node directory. The default root certificate is stored in the root-key.p12 in the configuration repository under the node directory.

When you federate a base application server, the following situations occur: the keystore and truststore are included, and the signer certificate is added to the deployment manager common truststore, which is located in the cell directory of the configuration repository.

All of the nodes put their signer certificates from the default root certificate in this common truststore (`trust.p12`). Additionally, after you federate a node, the default SSL configuration is automatically modified to point to the common truststore, which is located in the cell directory. The node can now communicate with all other servers in the cell.

All default SSL configurations contain a keystore with the name suffix DefaultKeyStore, a truststore with the name suffix DefaultTrustStore and a rootstore with the name suffix DefaultRootStore. These default suffixes instruct the WebSphere Application Server runtime to add the root signer of the personal certificate to the common truststore. If a truststore name does not end with DefaultKeyStore, the keystores root signer certificates are not added to the common truststore when you federate the server. You can change the default SSL configuration, but you must ensure that the correct trust is established for administrative connections, among others.

For more information, see “Default chained certificate configuration in SSL” on page 694 and “Web server plug-in default configuration in SSL” on page 820.

Certificate expiration monitoring

Certificate monitoring ensures that the default chained certificate for each node is not allowed to expire. The certificate expiration monitoring function issues a warning before certificates and signers are set to expire. Those certificates and signers that are located in keystores managed by the WebSphere Application Server configuration can be monitored. You can configure the expiration monitor to automatically replace a certificate. A chained certificate will be recreated based on the same data used for the initial creation and sign it with the same root certificate that signed the original certificate. A self-signed certificate or chained certificate is also recreated based upon the same data that is used for the initial creation.

The monitor also can automatically replace old signers with the signers from the new chained or self-signed certificates in keystores that are managed by WebSphere Application Server. The existing signer exchanges that occurred by the runtime during federation and by administration are preserved. For more information, see “Certificate expiration monitoring in SSL” on page 702.

The expiration monitor is configured to replace chained personal certificates that are signed by a root certificate in DmgrDefaultRootStore or NodeDefaultRootStore. The certificate is renewed using the same root certificate that was used to sign the original certificate.

The monitor also can automatically replace old signers with the signers from the new self-signed certificates in keystores that are managed by WebSphere Application Server. The existing signer exchanges that occurred by the runtime during federation and by administration are preserved. For more information, see “Certificate expiration monitoring in SSL” on page 702.

WebSphere Application Server clients: signer-exchange requirements

A new chained certificate is generated for each node during its initial startup. To ensure trust, clients must be given the root signers to establish a connection. The introduction of chained certificates in the current release makes this process simpler. Rather than exchanging the signer of a short lived self-signed certificate, you can exchange the long lived root signer which will allow for preserved trust across personal certificate renewals. In addition, you can gain access to the signer certificates of various nodes to which the client must connect with any one of the following options (for more information, see “Secure installation for client signer retrieval in SSL” on page 698):

- A signer exchange prompt enables you to import signer certificates that are not yet present in the truststores during a connection to a server. By default, this prompt is enabled for administrative

connections and can be enabled for any client SSL configuration. When this prompt is enabled, any connection that is made to a server where the signer is not already present offers the signer of the server along with the certificate information and a Secure Hash Algorithm (SHA) digest of the certificate for verification. The user is given a choice whether to accept these credentials. If the credentials are accepted, the signer is added to the truststore of the client until the signer is explicitly removed. The signer exchange prompt does not occur again when connecting to the same server unless the personal certificate changes.

Attention: It is unsafe to trust a signer exchange prompt without verifying the SHA digest. An unverified prompt can originate from a browser when a certificate is not trusted.

- You can run a retrieveSigners administrative script from a client prior to making connections to servers. To download signers, no administrative authority is required. To upload signers, you must have Administrator role authority. The script downloads all of the signers from a specified server truststore into the specified client truststore and can be called to download only a specific alias from a truststore. You can also call the script to upload signers to server truststores. When you select the CellDefaultTrustStore truststore as the specified server truststore and common truststore for a cell, all of the signers for that cell are downloaded to the specified client truststore, which is typically ClientDefaultTrustStore. For more information, see “retrieveSigners command” on page 700.
- You can physically distribute to clients the trust.p12 common truststore that is located in the cell directory of the configuration repository. When doing this distribution, however, you must ensure that the correct password has been specified in the `ssl.client.props` client SSL configuration file. The default password for this truststore is **`WebAS`**. Change the default password prior to distribution. Physical distribution is not as effective as the previous options. When changes are made to the personal certificates on the server, automated exchange can fail.

Dynamic SSL configuration changes

The SSL runtime for WebSphere Application Server maintains listeners for most SSL connections. A change to the SSL configuration causes the inbound connection listeners to create a new SSLContext object. Existing connections continue to use the current SSLContext object. Outbound connections automatically use the new configuration properties when they are attempted.

Make dynamic changes to the SSL configuration during off-peak hours to reduce the possibility of timing-related problems and to prevent the possibility of the server starting again. If you enable the runtime to accept dynamic changes, then change the SSL configuration and save the `security.xml` file. Your changes take effect when the new `security.xml` file reaches each node.

Note: If configuration changes cause SSL handshake failures, administrative connectivity failures also can occur, which can lead to outages. In this case, you must re-configure the SSL connections then perform manual node synchronization to correct the problem. You must carefully complete any dynamic changes. It is highly recommended that you perform changes to SSL configurations on a test environment prior to making the same changes to a production system. For more information, see “Dynamic configuration updates in SSL” on page 705.

Built-in certificate management

Certificate management that is comparable to iKeyMan functionality is now integrated into the keystore management panels of the administrative console. Use built-in certificate management to manage personal certificates, certificate requests, and signer certificates that are located in keystores. Additionally, you can remotely manage keystores. For example, you can manage a file-based keystore that is located outside the configuration repository on any node from the deployment manager. You also can remotely manage hardware cryptographic keystores from the deployment manager.

With built-in certificate management, you can replace a chained or self-signed certificate along with all of the signer certificates scattered across many truststores and retrieve a signer from a remote port by connecting to the remote SSL host and port and intercepting the signer during the handshake. The

certificate is first validated according to the certificate SHA digest, then the administrator must accept the validated certificate before it can be placed into a truststore.

When you make a certificate request, you can send it to a certificate authority (CA). When the certificate is returned, you can accept it within the administrative console. For more information, see “Certificate management in SSL” on page 708.

Tip: Although iKeyMan functionality still ships with WebSphere Application Server, configure keystores from the administrative console using the built-in certificate management functionality. iKeyMan is still an option when it is not convenient to use the administrative console. For more information, see Certificate management using iKeyman prior to SSL.

AdminTask configuration management

The SSL configuration management panels in the administrative console rely primarily on administrative tasks, which are maintained and enhanced to support the administrative console function. You can use wsadmin commands from a Java console prompt to automate the management of keystores, SSL configurations, certificates, and so on.

SSL configurations

Secure Sockets Layer (SSL) configurations contain attributes that enable you to control the behavior of both the client and the server SSL endpoints. You can assign SSL configurations to have specific management scopes. The scope that an SSL configuration inherits depends upon whether you create it using a cell, node, server, or endpoint link in the configuration topology.

When you create an SSL configuration, you can set the following SSL connection attributes:

- Keystore
- Default client certificate for outbound connections
- Default server certificate for inbound connections
- Truststore
- Key manager for selecting a certificate
- Trust manager or managers for establishing trust during the handshake
- Handshaking protocol
- Ciphers for negotiating the handshake
- Client authentication support and requirements

You can manage an SSL configuration using any of the following methods:

- Central management selection
- Direct reference selection
- Dynamic outbound connection selection
- Programmatic selection

Using the administrative console, you can manage all of the SSL configurations for WebSphere Application Server. From the administrative console, click **Security**

SSL certificates and key management > **Manage endpoint security configurations** > **Inbound** | **Outbound** > **SSL_configuration**. You can view an SSL configuration at the level it was created and in the inherited scope below that point in the topology. If you want the entire cell to view an SSL configuration, you must create the configuration at the cell level in the topology.

SSL configuration in the `security.xml` file

The attributes defining an SSL configuration repertoire entry for a specific management scope are stored in the `security.xml` file. The scope determines the point at which other levels in the cell topology can see the configuration, as shown in the following example:

```
<repertoire xmi:id="SSLConfig_1" alias="NodeDefaultSSLSettings"
managementScope="ManagementScope_1" type="JSSE">
<setting xmi:id="SecureSocketLayer_1" clientAuthentication="false"
clientAuthenticationSupported="false" securityLevel="HIGH" enabledCiphers=""
jsseProvider="IBMJSSE2" sslProtocol="SSL_TLS" keyStore="KeyStore_1"
trustStore="KeyStore_2" trustManager="TrustManager_1" keyManager="KeyManager_1"
clientKeyAlias="default" serverKeyAlias="default"/>
</repertoire>
```

The SSL configuration attributes from the previous code sample are described in Table 1.

Table 86. security.xml Attributes. This table lists the security.xml Attributes.

security.xml attribute	Description	Default	Associated SSL property
xmi:id	The xml:id attribute represents the unique identifier for this XML entry and determines how the SSL configuration is linked to other XML objects, such as SSLConfigGroup. This system-defined value must be unique.	The administrative configuration service defines the default value.	None. This value is used only for XML associations.
alias	The alias attribute defines the name of the SSL configuration. Direct selection uses the alias attribute and the node is not prefixed to the alias. Rather, the management scope takes care of ensuring that the name is unique within the scope.	The default is NodeDefaultSSLSettings.	com.ibm.ssl.alias
managementScope	The managementScope attribute defines the management scope for the SSL configuration and determines the visibility of the SSL configuration at runtime.	The default scope is the node.	The managementScope attribute is not mapped to an SSL property. However, it confirms whether or not the SSL configuration is associated with a process.
type	The type attribute defines the Java Secure Socket Extension (JSSE) or System Secure Sockets Layer (SSSL) configuration option. JSSE is the SSL configuration type for most secure communications within WebSphere Application Server.	The default is JSSE.	com.ibm.ssl.sslType
clientAuthentication	The clientAuthentication attribute determines whether SSL client authentication is required.	The default is false.	com.ibm.ssl.clientAuthentication
clientAuthenticationSupported	The clientAuthenticationSupported attribute determines whether SSL client authentication is supported. The client does not have to supply a client certificate if it does not have a client certificate. **Attention:** When you set the clientAuthentication attribute to `true`, you override the value that is set for the clientAuthenticationSupported attribute.	The default is false.	com.ibm.ssl.client.AuthenticationSupported

Table 86. security.xml Attributes (continued). This table lists the security.xml Attributes.

security.xml attribute	Description	Default	Associated SSL property
securityLevel	The securityLevel attribute determines the cipher suite group. Valid values include STRONG (128-bit ciphers), MEDIUM (40-bit ciphers), WEAK (for all ciphers without encryption), and CUSTOM (if the cipher suite group is customized. When you set the enabledCiphers attribute with a specific list of ciphers, the system ignores this attribute.	The default is STRONG.	com.ibm.ssl.securityLevel
enabledCiphers	You can set the enabledCiphers attribute to specify a unique list of cipher suites. Separate each cipher suite in the list with a space.	The default is the securityLevel attribute for cipher suite selection.	com.ibm.ssl.enabledCipherSuites
jsseProvider	The jsseProvider attribute defines a specific JSSE provider.	The default is IBMJSSE2.	com.ibm.ssl.contextProvider
sslProtocol	The sslProtocol attribute defines the SSL handshake protocol. Valid options include: SSL_TLS - which is SSLv3 and TLSv1 SSL - which is SSLv3 SSLv2 SSLv3 TLS - which is TLSv1 TLSv1 SSL_TLSv2 - which is SSLv3 and TLSv1, TLSv1.1, TLSv1.2 TLSv1.1 TLSv1.2 The listSSLProtocols command provides more information about which protocol are valid in particular configurations, such as FIPS 140-2 or SP800-131.	The default is SSL_TLS.	com.ibm.ssl.protocol
keyStore	The keyStore attribute defines the keystore and attributes of the keyStore instance that the SSL configuration uses for key selection.	The default is NodeDefaultKeyStore.	For more information, see Keystore configurations.
trustStore	The trustStore attribute defines the key store that the SSL configuration uses for certificate signing verification.	The default is NodeDefaultTrustStore.	A trustStore is a logical JSSE term. It signifies a key store that contains signer certificates. Signer certificates validate certificates that are sent to WebSphere Application Server during an SSL handshake.

Table 86. security.xml Attributes (continued). This table lists the security.xml Attributes.

security.xml attribute	Description	Default	Associated SSL property
keyManager	The keyManager attribute defines the key manager that WebSphere Application Server uses to select keys from a key store. A JSSE key manager controls the javax.net.ssl.X509KeyManager interface. A custom key manager controls the javax.net.ssl.X509KeyManager and the com.ibm.wsspi.ssl.KeyManagerExtendedInfo interfaces. The com.ibm.wsspi.ssl.KeyManagerExtendedInfo interface provides more information from WebSphere Application Server.	The default is IbmX509.	com.ibm.ssl.keyManager defines a well-known key manager and accepts the algorithm and algorithm|provider formats, for example IbmX509 and IbmX509|IBMJSSE2. com.ibm.ssl.customKeyManager defines a custom key manager and takes precedence over the other keyManager properties. This class must implement javax.net.ssl.X509KeyManager and can implement com.ibm.wsspi.ssl.KeyManagerExtendedInfo. For more information, see "Key manager control of X.509 certificate identities" on page 679
trustManager	The trustManager determines which trust manager or list of trust managers to use for determining whether to trust the peer side of the connection. A JSSE trust manager implements the javax.net.ssl.X509TrustManager interface. A custom trust manager might also implement com.ibm.wsspi.ssl.TrustManagerExtendedInfo interface to get more information from the WebSphere Application Server environment.	The default is IbmPKIX, which can be configured for certificate revocation list (CRL) verification when the certificate contains a CRL distribution point. The other option is IbmX509.	com.ibm.ssl.trustManager defines a well-known trust manager, which is required for most handshake situations. com.ibm.ssl.trustManager performs certificate expiration checking and signature validation. You can define com.ibm.ssl.customTrustManagers with additional custom trust managers that are called during an SSL handshake. Separate additional trust managers with the vertical bar (|) character. For more information, see "Trust manager control of X.509 certificate trust decisions"

Client SSL configurations are managed using the `ssl.client.props properties` file. The `ssl.client.props` file is located in the ${USER_INSTALL_ROOT}/properties directory for each profile. For more information about configuring this file, see the "ssl.client.props client configuration file" on page 755. Specifying any javax.net.ssl system properties will override the corresponding property in the `ssl.client.props` file.

Trust manager control of X.509 certificate trust decisions

The role of the trust manager is to validate the Secure Sockets Layer (SSL) certificate that is sent by the peer, which includes verifying the signature and checking the expiration date of the certificate. A Java Secure Socket Extension (JSSE) trust manager determines if the remote peer can be trusted during an SSL handshake.

WebSphere Application Server has the ability to call multiple trust managers during an SSL connection. The default trust manager does the standard certificate validation; custom trust manager plug-ins run customized validation such as host name verification. For more information, see "Example: Developing a custom trust manager for custom SSL trust decisions" on page 737

When a trust manager is configured in a server-side SSL configuration, the server calls the isClientTrusted method. When a trust manager is configured in a client-side SSL configuration, the client calls the isServerTrusted method. The peer certificate chain is passed to these methods. If the trust manager chooses not to trust the peer information, it might produce an exception to force a handshake failure.

Optionally, WebSphere Application Server provides the com.ibm.wsspi.ssl.TrustManagerExtendedInfo interface so that additional information can be passed to the trust manager. For more information, see the com.ibm.wsspi.ssl.TrustManagerExtendedInfo interface.

Default IbmX509 trust manager

The default IbmX509 trust manager, which is used in the following code sample, establishes trust by performing standard certificate validation.

```
<trustManagers xmi:id="TrustManager_1132357815717" name="IbmX509" provider="IBMJSSE2"
algorithm="IbmX509" managementScope="ManagementScope_1132357815717"/>
```

The trust manager provides a signer certificate to verify the peer certificate that is sent during the handshake. The signers who are added to the truststore for the SSL configuration must be trustworthy. If you do not trust the signers or do not want to allow others to connect to your servers, consider removing default root certificates from certificate authorities (CA). You might also remove any certificates if you cannot verify their origination.

Default IbmPKIX trust manager

You can use the default IbmPKIX trust manager to replace the IbmX509 trust manager, which is shown in the following code sample:

```
<trustManagers xmi:id="TrustManager_1132357815719" name="IbmPKIX" provider="IBMJSSE2"
algorithm="IbmPKIX" trustManagerClass="" managementScope="ManagementScope_1132357815717">
<additionalTrustManagerAttrs xmi:id="DescriptiveProperty_1132357815717"
name="com.ibm.security.enableCRLDP" value="true" type="boolean"/>
<additionalTrustManagerAttrs xmi:id="DescriptiveProperty_1132357815718"
name="com.ibm.jsse2.checkRevocation" value="true" type="boolean"/>
  </trustManagers>

<trustManagers xmi:id="TrustManager_managementNode_2" name="IbmPKIX" provider=
"IBMJSSE2" algorithm="IbmPKIX" trustManagerClass=""
managementScope="ManagementScope_managementNode_1">
<additionalTrustManagerAttrs xmi:id="DescriptiveProperty_1" name="com.ibm.se
curity.enableCRLDP" value="false" type="boolean" displayNameKey="" nlsRangeKey="
" hoverHelpKey="" range="" inclusive="false" firstClass="false"/>
<additionalTrustManagerAttrs xmi:id="DescriptiveProperty_2" name="com.ibm.js
se2.checkRevocation" value="false" type="boolean" displayNameKey="" nlsRangeKey=
"" hoverHelpKey="" range="" inclusive="false" firstClass="false"/>
<additionalTrustManagerAttrs xmi:id="DescriptiveProperty_3" name="ocsp.enabl
e" value="false" type="String" displayNameKey="" nlsRangeKey="" hoverHelpKey=""
range="" inclusive="false" firstClass="false"/>
<additionalTrustManagerAttrs xmi:id="DescriptiveProperty_4" name="ocsp.respo
nderURL" value="http://ocsp.example.net:80" type="String" displayNameKey=""
nlsRangeKey="" hoverHelpKey="" range="" inclusive="false" firstClass="false"/>
<additionalTrustManagerAttrs xmi:id="DescriptiveProperty_5" name="ocsp.respo
nderCertSubjectName" value="" type="String" displayNameKey="" nlsRangeKey="" hov
erHelpKey="" range="" inclusive="false" firstClass="false"/>
<additionalTrustManagerAttrs xmi:id="DescriptiveProperty_6" name="ocsp.respo
nderCertIssuerName" value="" type="String" displayNameKey="" nlsRangeKey="" hove
rHelpKey="" range="" inclusive="false" firstClass="false"/>
<additionalTrustManagerAttrs xmi:id="DescriptiveProperty_7" name="ocsp.respo
nderCertSerialNumber" value="" type="String" displayNameKey="" nlsRangeKey="" ho
verHelpKey="" range="" inclusive="false" firstClass="false"/>
</trustManagers>
```

See “Example: Enabling certificate revocation checking with the default IbmPKIX trust manager” on page 680 for additional information in using the default IbmPKIX trust manager.

In addition to its role of standard certificate verification, the IbmPKIX trust manager checks for OCSP properties and for certificates that contain certificate revocation list (CRL) distribution points. This process is known as extended CRL checking. When you select a trust manager, its associated properties are automatically set as Java System properties so that the IBMCertPath and IBMJSSE2 providers are aware that CRL checking is enabled.

Differences between the Ibmx509 and the IbmPKIX trust managers

x.509 certificate validation requirements are more stringent in the IbmX509 trustmanager than in the IbmPKIX trustmanager. For example:

- The IbmX509 trustmanager validates the entire certificate chain regardless of which certificates the client/server trusts. However, the IbmPKIX trustmanager does not validate a certificate even if you tell the IbmPKIX trustmanager that you want to trust that certificate. The IbmPKIX trustmanager only validates the certificates from the one signed by the certificate you trust, to the leaf certificate. Also,

- The IbmX509 requires that any root CA certificate must possess the BASIC CONSTRAINTS extension. Otherwise the certificate cannot be used as a root CA certificate. IbmPKIX does not have this BASIC CONSTRAINTS requirement for root CA certificates.
- The IbmX509 trust manager performs signature validation and certificate expiration checks to validates certificates. . The IbmPKIX trust manager performs these same validations, plus more advanced Certificate Revocation List checking (CRL), which determines if the certificate authority (CA) has revoked the certificate.
- The IbmPKIX trust manager automatically obtains a CRL when a certificate is received if CRL distribution point (DP) extensions exist.

Additionally, the Online Certificate Status Protocol (OCSP) can be used to perform an online check of certificate validity. However this capability requires you to set additional system properties, as documented in the *Java Certification Path API Programmer's Guide*, which is available on the IBM developerWorks web site.

Custom trust manager

You can define a custom trust manager to perform additional trust checking, which is based upon the needs of the environment. For example, in one environment, you might enable connections from the same Transmission Control Protocol (TCP) subnet only. The com.ibm.wsspi.ssl.TrustManagerExtendedInfo interface provides extended information about the connection that is not provided by the standard Java Secure Sockets Extension (JSSE) javax.net.ssl.X509TrustManager interface. The configured trustManagerClass attribute determines which class is instantiated by the runtime, as shown in the following code sample:

```
<trustManagers xmi:id="TrustManager_1132357815718" name="CustomTrustManager"
trustManagerClass="com.ibm.ws.ssl.core.CustomTrustManager"
managementScope="ManagementScope_1132357815717"/>
```

The trustManagerClass attribute must implement the javax.net.ssl.X509TrustManager interface and, optionally, can implement the com.ibm.wsspi.ssl.TrustManagerExtendedInfo interface.

Disabling the default trust manager

In some cases, you might not want to perform the standard certificate verification that is provided by the IbmX509 and IbmPKIX default trust managers. For example, you might be working with an internal automated test infrastructure that is not concerned with SSL client or server authentication, integrity, or confidentiality. The following sample code shows a basic custom trust manager such as com.ibm.ws.ssl.core.CustomTrustManager whose property is set to `true`.

```
com.ibm.ssl.skipDefaultTrustManagerWhenCustomDefined=true
```

You can set this property in the global properties at the top of the `ssl.client.props` file for clients or in the `security.xml` custom properties file for servers. You must configure a custom trust manager when you disable the default trust manager to prevent the server from calling the default trust manager even though it is configured. Disabling the default trust manager is not a common practice. Be sure to test the system with the disabled default trust manager in a test environment first. For more information on setting up a custom trust manager, see “Creating a custom trust manager configuration for SSL” on page 733

Key manager control of X.509 certificate identities

The role of a Java Secure Socket Extension (JSSE) key manager is to retrieve the certificate that is used to identify the client or server during a Secure Sockets Layer (SSL) handshake.

WebSphere Application Server provides a default key manager that can select a certificate from a keystore when you define the following SSL configuration properties:

com.ibm.ssl.keyStoreClientAlias
: Defines the alias that is chosen from the keystore for the client side of a connection. This alias must be present in the keystore.

com.ibm.ssl.keyStoreServerAlias
Defines the alias that is chosen from the keystore for the server side of a connection. This alias must be present in the keystore.

These two properties are set automatically when you use the administrative console because the default key manager is already configured.

With WebSphere Application Server, you can configure only one key manager at a time for a given SSL configuration. If you want custom certificate selection logic on the client side, you must write a new custom key manager. The custom key manager could provide function that prompts the user to choose a certificate dynamically. Also, you can implement an extended interface so that a key manager can provide information during connection time. For more information on the extended interface, see the com.ibm.wsspi.ssl.KeyManagerExtendedInfo interface. For more information on custom key manager development, see Creating a custom key manager for SSL.

Default IbmX509 key manager

The default IbmX509 key manager chooses a certificate to serve as the identity for an SSL handshake. The key manager is called to enable client authentication on either side of the SSL handshake; frequently on the server-side, and less frequently on the client side according to client and server requirements. If a keystore is not configured on the client-side and SSL client authentication is enabled, the key manager cannot select a certificate to send to the server. Therefore, the handshake fails.

The following sample code shows the key manager configuration in the `security.xml` file for an IbmX509 key manager.

```
<keyManagers xmi:id="KeyManager_1" name="IbmX509"
provider="IBMJSSE2" algorithm="IbmX509" keyManagerClass=""
managementScope="ManagementScope_1"/>
```

You do not specify the keyManagerClass class because the key manager is provided by the IBMJSSE2 provider. However, you can specify whether the key manager is a custom class implementation, in which case you must specify the keyManager class, or an algorithm name that WebSphere Application Server can start from the Java security provider framework.

Custom key manager

The following sample code shows the key manager configuration in the `security.xml` file for a custom class.

```
<keyManagers xmi:id="KeyManager_2" name="CustomKeyManager"
keyManagerClass="com.ibm.ws.ssl.core.CustomKeyManager"
managementScope="ManagementScope_1"/>
```

The custom class must implement the javax.net.ssl.X509KeyManager interface and, optionally, implement the com.ibm.wsspi.ssl.KeyManagerExtendedInfo interface to retrieve additional WebSphere Application Server information. This interface replaces the function of the default key manager because you can configure only one key manager at a time. Therefore, the custom key manager has sole responsibility for selecting the alias to use from the configured keystore. The benefit of a custom key manager is its ability, on the client side, to prompt for an alias. This process enables the user to decide which certificate to use in situations where the user knows the client certificate identity. For more information, see Creating a custom key manager for SSL.

Example: Enabling certificate revocation checking with the default IbmPKIX trust manager

The IbmPKIX trust manager is enabled in the WebSphere Application Server by default. The IbmPKIX trust manager allows certificate revocation checking to occur. You enable certificate revocation checking by using the administrative console or by manually updating the `ssl.client.props` file.

The default IbmPKIX trust manager

The IbmPKIX trust manager is enabled by default, but revocation checking is not enabled by default. The following trust manager definition for IbmPKIX reflects the default condition:

```
<trustManagers xmi:id="TrustManager_managementNode_2" name="IbmPKIX" provider=
"IBMJSSE2" algorithm="IbmPKIX" trustManagerClass=""
managementScope="ManagementScope_managementNode_1">
<additionalTrustManagerAttrs xmi:id="DescriptiveProperty_1" name="com.ibm.se
curity.enableCRLDP" value="false" type="boolean" displayNameKey="" nlsRangeKey="
" hoverHelpKey="" range="" inclusive="false" firstClass="false"/>
<additionalTrustManagerAttrs xmi:id="DescriptiveProperty_2" name="com.ibm.js
se2.checkRevocation" value="false" type="boolean" displayNameKey="" nlsRangeKey=
"" hoverHelpKey="" range="" inclusive="false" firstClass="false"/>
<additionalTrustManagerAttrs xmi:id="DescriptiveProperty_3" name="ocsp.enable
e" value="false" type="String" displayNameKey="" nlsRangeKey="" hoverHelpKey=""
range="" inclusive="false" firstClass="false"/>
<additionalTrustManagerAttrs xmi:id="DescriptiveProperty_4" name="ocsp.respo
nderURL" value="http://ocsp.example.net:80" type="String" displayNameKey=""
nlsRangeKey="" hoverHelpKey="" range="" inclusive="false" firstClass="false"/>
<additionalTrustManagerAttrs xmi:id="DescriptiveProperty_5" name="ocsp.respo
nderCertSubjectName" value="" type="String" displayNameKey="" nlsRangeKey="" hov
erHelpKey="" range="" inclusive="false" firstClass="false"/>
<additionalTrustManagerAttrs xmi:id="DescriptiveProperty_6" name="ocsp.respo
nderCertIssuerName" value="" type="String" displayNameKey="" nlsRangeKey="" hove
rHelpKey="" range="" inclusive="false" firstClass="false"/>
<additionalTrustManagerAttrs xmi:id="DescriptiveProperty_7" name="ocsp.respo
nderCertSerialNumber" value="" type="String" displayNameKey="" nlsRangeKey="" ho
verHelpKey="" range="" inclusive="false" firstClass="false"/>
</trustManagers>
```

Enabling certificate revocation checking with the default IbmPKIX trust manager

You can view and change IbmPKIX Trust Manager Custom Properties using the administrative console.

To do this,

- Click **Security > SSL certificate and key management**.
- Under Related Items, click **Trust managers**.
- Click **IbmPKIX**.
- Under Additional Properties, click **Custom properties**.

IbmPKIX custom properties

com.ibm.jsse2.checkRevocation

This property configures revocation checking for the Java Virtual Machine (JVM). This property is set to false by default because the default WebSphere certificates used for SSL communication do not contain certificate revocation list (CRL) distribution points or Online Certificate Status Protocol (OCSP) information.

Note: Since this property is a JVM property, this value is in effect for the entire application server. If the property is defined in trust managers at different scopes, the value in effect is used from the most specifically scoped IbmPKIX trust manager. For example, the property for an IbmPKIX trust manager defined at the node level overrides the property for an IbmPKIX trust manager defined at the cell level. This property is ignored for the IbmX509 trust manager.

default
false

com.ibm.security.enableCRLDP

This property configures CRL distribution point checking for the PKIX trust manager.

Note: If you enable CRL distribution point revocation checking, the certificates used for secure sockets layer (SSL) must contain a valid distribution point and the distribution point must be accessible or else SSL communication will fail and the server will not function correctly.

default
false

For certificates that do not contain an internal CRL distribution point, the following properties can used so the revocation status will be checked against a remote LDAP server containing the CRL.

com.ibm.security.ldap.certstore.host
This property specifies the LDAP server host name containing trusted certificates or certificate revocation lists. The target LDAP server host is used to obtain CA certificates or certificate revocation lists when validating a certificate and the local truststore does not contain the required certificate. The local truststore must contain the required certificates if an LDAP server is not specified. In cases when an LDAP server is used, the root CA certificates must also be located in the local truststore as the LDAP server is not a trusted certificate store.

Note: Enabling this property in addition to the com.ibm.jsse2.checkRevocation property enables revocation checking. The remote LDAP server must contain a valid certificate revocation list and the server must be accessible. If the revocation status cannot be determined then the check will fail and SSL communication will fail and the server will not function correctly.

default
none

com.ibm.security.ldap.certstore.port
This property specifies the LDAP server port. A port value of 389 will be used by default if no LDAP server port is specified.

default
389

The following Java Development Kit (JDK) properties apply to enabling certificate revocation checking with the default IbmPKIX trust manager:

- ocsp.enable
- ocsp.responder
- ocsp.responderCertSubjectName
- ocsp.responderCertIssuerName
- ocsp.responderCertSerialNumber

These JDK properties can be set using the administrative console. You should reference Java(TM) Certification Path API Programmer's Guide - SDK 6.0 for descriptions of these properties and their allowable settings.

Note: In addition to its role of standard certificate verification, the IbmPKIX trust manager checks for certificates that contain CRL distribution points. This process is known as extended CRL checking. By default, CRL distribution point revocation checking is disabled. To enable CRL distribution point revocation checking, you must set the following properties to `true` using the administrative console:

- com.ibm.security.enableCRLDP
- com.ibm.jsse2.checkRevocation

OCSP properties and CRL properties affect certificate revocation checking. By default OCSP properties are checked first. If there is an error validating the certificate with OCSP, then validation uses a CRL distribution point instead.

When you select a trust manager, its associated properties are automatically set as Java system properties so that the IBMCertPath and IBMJSSE2 providers are aware that CRL checking is enabled or disabled. Similarly, the same applies for OCSP properties, which are java.security.Security properties.

Client considerations

You can also enable revocation checking for WebSphere application and administrative clients by directly setting the properties in the `ssl.client.props` file. An example of the `ssl.client.props` file follows:

```
#-------------------------------------------------------------------------
# Default Revocation Checking Properties
# These properties are used for certificate revocation checking with the IBM
# PKIX TrustManager.
#
# To enable CRL Distribution Points extension checking, use the system property
# com.ibm.security.enableCRLDP.
#
# OCSP checking is not enabled by default. It is enabled by setting the
# ocsp.enable property to "true".  Use of the other ocsp properties is optional.
#
# Note: Both OCSP and CRLDP checking is only effective if revocation checking
# has also been enabled by setting com.ibm.jsse2.checkRevocation to "true".
#
#-------------------------------------------------------------------------
com.ibm.jsse2.checkRevocation=false
com.ibm.security.enableCRLDP=false
#ocsp.enable=true
#ocsp.responderURL=http://ocsp.example.net
#ocsp.responderCertSubjectName=CN=OCSP Responder, O=XYZ Corp
#ocsp.responderCertIssuerName=CN=Enterprise CA, O=XYZ Corp
#ocsp.responderCertSerialNumber=2A:FF:00
```

Note: In order for these properties to be effective, you must ensure that the IbmPKIX trust manager is initialized by setting `com.ibm.ssl.trustManager=IbmPKIX`.

In addition, for revocation checking to be processed successfully on the client, you are required to turn off the signer exchange prompt. To do this, change the value of the **com.ibm.ssl.enableSignerExchangePrompt** property to `false`, in the ssl.client.props file.

For more information on these properties, see Java(TM) Certification Path API Programmer's Guide - SDK 6.0.

Keystore configurations for SSL

Use keystore configurations to define how the runtime for WebSphere Application Server loads and manages keystore types for Secure Sockets Layer (SSL) configurations.

By default, the java.security.Security.getAlgorithms("KeyStore") attribute does not display a predefined list of keystore types in the administrative console. Instead, WebSphere Application Server retrieves all of the KeyStore types that can be referenced by the java.security.KeyStore object, including hardware cryptographic, z/OS platform, IBM i platform, IBM Java Cryptography Extension (IBMJCE), and Java-based content management system (CMS)-provider keystores. If you specify a keystore provider in the `java.security` file or add it to the provider list programmatically, WebSphere Application Sever also retrieves custom keystores. The retrieval list depends upon the java.security configuration for that platform and process.

IBMJCE file-based keystores (JCEKS, JKS, and PKCS12)

A typical IBMJCE file-based keystore configuration is shown in the following sample code:

```
<keyStores xmi:id="KeyStore_1" name="NodeDefaultKeyStore"
password="{xor}349dkckdd=" provider="IBMJCE"
location="${USER_INSTALL_ROOT}/config/cells/myhostNode01Cell
/nodes/myhostNode01/key.p12" type="PKCS12" fileBased="true"
hostList="" initializeAtStartup="true" readOnly="false"
description="Default key store for myhostNode01" usage="SSLKeys"
managementScope="ManagementScope_1"/>
```

For more information about default keystore configurations, see Default chained certificate configuration in SSL.

Table 1 describes the attributes that are used in the sample code.

Table 87. keystore configurations. This table describes the keystore configurations.

Attribute name	Default	Description
xmi:id	Varies	A value that issued to reference the keystore from another area in the configuration, for example, from an SSL configuration. Make this value unique within the `security.xml` file.
name	For Java Secure Socket Extension (JSSE) keystore: NodeDefaultKeyStore. For JSSE truststore: NodeDefaultTrustStore.	A name that is used to identify the keystore by sight. The name can determine if the keystore is a default keystore based upon whether the name ends with DefaultKeyStore or DefaultTrustStore.
password	The default keystore password is **WebAS**. It is recommended that this be changed as soon as possible. See Updating default key store passwords using scripting for more information.	The password that is used to access the keystore name is also the default that is used to store keys within the keystore.
description	No default	A description of the keystore.
usage	An attribute specifying what the keystore is used for.	Valid values are: SSLKeys, KeySetKeys, RootKeys, DeletedKeys, DefaultSigners, RSATokenKeys.
provider	The default provider is IBMJCE.	The Java provider that implements the type attribute (for example, PKCS12 type). The provider can be unspecified and the first provider that implements the keystore type specified is used.
location	The default varies, but typically references a `key.p12` file or a `trust.p12` file in the node or cell directories of the configuration repository. These files are PKCS12 type keystores.	The keystore location reference. If the keystore is file-based, the location can reference any path in the file system of the node where the keystore is located. However, if the location is outside of the configuration repository, and you want to manage the keystore remotely from the administrative console or from the `wsadamin` utility, then specify the hostList attribute that contains the host name of the node where it resides.
type	The default Java crypto device keystore type is `PKCS12`.	This type specifies the keystore. Valid types can be those returned by the java.security.Security.getAlgorithms("KeyStore") attribute. These types include the following keystore types, and availability depends on the process and platform `java.security` configuration: • JKS • JCEKS • PKCS12 • PKCS11 (Java crypto device) • CMSKS • IBMi5OSKeyStore • JCERACFKS • JCECCAKS keystores (replacing JCE4758KS) - (z/OS crypto device)
fileBased	The default is `true`.	This option is required for default keystores. It indicates a file-system keystore so you can use a FileInputStream or FileOutputStream for loading and storing the keystore.
hostList	The hostList attribute is used to specify a remote hostname so that the keystore can be remotely managed. There are no remotely managed keystores by default. All default keystores are managed locally in the configuration repository and synchronized out to each of the nodes.	The option manages a keystore remotely. You can set the host name of a valid node for a keystore. When you use either the administrative console or the wsadmin utility to manage certificates for this keystore, an MBean call is made to the node where the keystore exists for the approved operation. You can specify multiple hosts, although synchronization of the keystore operations are not guaranteed. For example, one of the hosts that is listed might be down when a specific operation is performed. Therefore, use multiple hosts in this list.
initializeAtStartup	The default is `true`.	This option informs the runtime to initialize the keystore during startup. This option can be important for hardware cryptographic device acceleration.

Table 87. keystore configurations (continued). This table describes the keystore configurations.

Attribute name	Default	Description
readOnly	The default is `false`.	This option informs the configuration that you cannot write to this keystore. That is, certain update operations on the keystore cannot be attempted and are not allowed. An example of a read-only keystore type is JCERACFKS on the z/OS platform. This type is read-only from the WebSphere certificate management standpoint, but you can also update it using the keystore management facility for RACF.
managementScope	The default scope is the node scope for a base Application Server environment and the cell scope for a Network Deployment environment.	This option references a particular management scope in which you can see this keystore. For example, if a hardware cryptographic device is physically located on a specific node, then create the keystore from a link to that node in the topology view under **Security** > **Security Communications** > **SSL configurations**. You can also use management scope to isolate a keystore reference. In some cases, you might need to allow only a specific application server to reference the keystore; the management scope is for that specific server.

CMS keystores

You can set some provider-specific attributes in CMS keystores.

If the CMSKS provider supports the createStashFileForCMS attribute, and you set the attribute to `true` for CMSKS keystores, WebSphere Application Server creates an `.sth` file in the keystore location that is referenced by the attribute. The `.sth` extension is appended to the keystore name. For example, if the CMSKS keystore is available for a plug-in configuration and you set createStashFileForCMS to `true`, the stash file that is represented in the following sample code is created in the `${USER_INSTALL_ROOT}\profiles\AppSrv01/config/cells/myhostCell01/nodes/myhostNode01/servers/webserver1/plugin-key.sth` path.

```
<keyStores xmi:id="KeyStore_1132071489571" name="CMSKeyStore"
password="{xor}HRYNFAtrbxEwOzpvbhw6MzM=" provider="IBMCMSProvider"
location="${USER_INSTALL_ROOT}\profiles\AppSrv01/config/cells/myhostCell01
/nodes/myhostNode01/servers/webserver1/plugin-key.kdb" type="CMSKS"
fileBased="true" createStashFileForCMS="true"
managementScope="ManagementScope_1132071489569"/>
```

When you create a CMS keystore, the CMS provider is **IBMi5OSJSSEProvider**, and the CMS type is **IBMi5OSKeyStore**, as shown in the following sample code:

```
<keyStores xmi:id="KeyStore_1132071489571" name="CMSKeyStore"
password="{xor}HRYNFAtrbxEwOzpvbhw6MzM=" provider="IBMi5OSJSSEProvider"
location="${USER_INSTALL_ROOT}\profiles\AppSrv01/config/cells/myhostCell01
/nodes/myhostNode01/servers/webserver1/plugin-key.kdb" type="IBMi5OSKeyStore"
fileBased="true" createStashFileForCMS="true"
managementScope="ManagementScope_1132071489569"/>
```

Hardware cryptographic keystores

For cryptographic device configuration, see “Key management for cryptographic uses” on page 807.

You can add a slot either as the custom property, `com.ibm.ssl.keyStoreSlot`, or as the configuration attribute, **slot="0"**. The custom property is read before the attribute for backwards compatibility.

Dynamic outbound selection of Secure Sockets Layer configurations

WebSphere Application Server provides dynamic outbound selection that enables you to choose a specific Secure Sockets Layer (SSL) configuration and certificate alias for each outbound protocol, target host, target port, or any combination of these attributes. You can specify the dynamic selection information for outbound connections from a pure client or from a server that is acting as a client.

Before the SSL runtime for WebSphere Application Server starts an outbound connection, the runtime attempts to match the outbound protocol, target host, and target port attributes with the dynamic outbound selection information that is associated with an SSL configuration and certificate alias in the configuration.

The runtime caches both selection misses and selection hits, so the impact on performance can be minimal. However, a relationship exists between the amount of dynamic outbound selection information and its impact on the initial connection performance.

Target information during outbound connections

The dynamic outbound selection configurations are only effective when the outbound protocol uses the JSSEHelper application programming interface (API) when you select an SSL configuration with a specified connectionInfo hash map. This hash map must contain the following properties:

com.ibm.ssl.direction
The value for outbound connections is OUTBOUND.

com.ibm.ssl.remoteHost
The format should match what the protocol provides. Typically this is the canonical Domain Name Space (DNS), but it also could be the IP address.

gotcha: The name comparison is performed as a case-insensitive comparison. There is no name resolution processing performed during the string comparison.

com.ibm.ssl.remotePort
The port is target port.

com.ibm.ssl.endPointName
The value for an outbound connection must be one of the following protocol strings:

- IIOP
- HTTP
- SIP
- LDAP
- ADMIN_IPC
- ADMIN_SOAP
- BUS_TO_BUS
- BUS_CLIENT
- BUS_TO_WEBSPHERE_MQ
- WEBSPHERE_MQ_CLIENT

Central management of SSL configurations

By default, Secure Sockets Layer (SSL) configurations for servers are managed from a central location in the topology view of the administrative console. You can associate an SSL configuration and certificate alias with a specific management scope. This method is the most efficient method to manipulate and modify configurations when the server topology changes.

In prior releases, SSL configurations are managed for each process. You have to maintain individual settings for each SSL configuration in the topology. In this release of WebSphere Application Server, management control of your SSL configurations offers more options and additional flexibility. You are able to make coarse-grained changes for the entire topology using the cell-scope and also make fine-grained changes using a particular endpoint name for a specific application server process. Because the SSL configuration associations manifest an inheritance behavior, you can simplify the number of associations by referencing only the highest level management scope that needs a unique configuration.

The topology view provides the scoping mechanism. The SSL configuration inherits its scope, which can be seen as its display in the topology. The scope encompasses the level where you created the configuration and all the subsequent levels that point. For example, when you create an SSL configuration

at a specific node, that configuration can be seen by that node agent and by every application server that is part of that node. Any application server or node that is not part of this particular node can not see this SSL configuration.

Your security environment influences issues such as the uniqueness of the SSL configurations, as well as the SSL configuration and the certificate alias placement in the topology. You are also able to configure different certificate aliases and different SSL configurations for inbound connections versus outbound connections.

To configure the inbound and outbound topologies, which must be done separately in the administrative console, click **Security** > **SSL certificates and key management** > **Manage endpoint security configurations** > **Inbound | Outbound**.

Default centrally managed SSL configuration

The default management scope is the node scope. When a node is federated into a cell, the default SSL configurations for the node are maintained, as shown in the following sample code for the sslConfigGroups and management scopes attributes:

```
<sslConfigGroups xmi:id="SSLConfigGroup_1" name="myhostNode01"
direction="inbound" certificateAlias="default" sslConfig="SSLConfig_1"
managementScope="ManagementScope_1"/>
<sslConfigGroups xmi:id="SSLConfigGroup_2" name="myhostNode01"
direction="outbound" certificateAlias="default" sslConfig="SSLConfig_1"
managementScope="ManagementScope_1"/>

<managementScopes xmi:id="ManagementScope_1"
scopeName="(cell):myhostNode01Cell:(node):myhostNode01" scopeType="node"/>
```

The SSL configuration xmi:id "SSLConfig_1" is also federated and applicable:

```
<repertoire xmi:id="SSLConfig_1" alias="NodeDefaultSSLSettings"
managementScope="ManagementScope_1">
<setting xmi:id="SecureSocketLayer_1" clientAuthentication="true"
securityLevel="HIGH" enabledCiphers="" jsseProvider="IBMJSSE2"
sslProtocol="SSL_TLS" keyStore="KeyStore_1" trustStore="KeyStore_2"
trustManager="TrustManager_1" keyManager="KeyManager_1"/>
</repertoire>
```

The keystores that are associated with the SSLConfig_1 SSL configuration are also federated, and `key.p12` is located in the node directory of the configuration repository:

```
<keyStores xmi:id="KeyStore_1" name="NodeDefaultKeyStore"
password="{xor}HRYNFAtrbxEwOzpvbhw6MzM=" provider="IBMJCE"
location="${USER_INSTALL_ROOT}/config/cells/myhostNode01Cell/nodes
/myhostNode01/key.p12" type="PKCS12" fileBased="true" hostList=""
initializeAtStartup="true" managementScope="ManagementScope_1"/>
<keyStores xmi:id="KeyStore_2" name="NodeDefaultTrustStore"
password="{xor}HRYNFAtrbxEwOzpvbhw6MzM=" provider="IBMJCE"
location="${USER_INSTALL_ROOT}/config/cells/myhostNode01Cell
/nodes/myhostNode01/trust.p12" type="PKCS12" fileBased="true"
hostList="" initializeAtStartup="true" managementScope="ManagementScope_1"/>
```

Secure Sockets Layer node, application server, and cluster isolation

Secure Sockets Layer (SSL) enables you to ensure that any client that attempts to connect to a server during the handshake first performs server authentication. Using SSL configurations at the node, application server, and cluster scopes, you can isolate communication between severs that should not be allowed to communicate with each other over secure ports.

Before you attempt to isolate communications controlled by WebSphere Application Server, you must have a good understanding of the deployment topology and application environment. To isolate a node, application server, or cluster, you must be able to control the signers that are contained in the truststores

that are associated with the SSL configuration. When the client does not contain the server signer, it cannot establish a connection to the server. By default, WebSphere uses chained certificates and each node has a unique root certificate signer. Because they the node shares the same root signer, all of the server in that node can connect to each other because they share the same root signer. However, if you use self-signed certificates, the server that created the personal certificate controls the signer, although you do have to manage the self-signed certificates. If you obtain certificates from a certificate authority (CA), you must obtain multiple CA signers because all of the servers can connect to each other if they share the same signer.

Authenticating only the server-side of a connection is not adequate protection when you need to isolate a server. Any client can obtain a signer certificate for the server and add it to its trust store. SSL client authentication must also be enabled between servers so that the server can control its connections by deciding which client certificates it can trust. For more information, see Enabling Secure Sockets Layer client authentication for a specific inbound endpoint, which applies as well to enabling SSL client authentication at the cell level.

Isolation also requires that you use centrally managed SSL configurations for all or most endpoints in the cell. Centrally managed configurations can be scoped, unlike direct or end point configuration selection, and they enable you to create SSL configurations, key stores, and trust stores at a particular scope. Because of the inheritance hierarchy of WebSphere Application Server cells, if you select only the properties that you need for an SSL configuration, only these properties are defined at your selected scope or lower. For example, if you configure at the node scope, your configuration applies to the application server and individual end point scopes following the node scope. For more information, see Associating Secure Sockets Layer configurations centrally with inbound and outbound scopes, Selecting an SSL configuration alias directly from an endpoint configuration, and Associating a Secure Sockets Layer configuration dynamically with an outbound protocol and remote secure endpoint

When you configure the key stores, which contain cryptographic keys, you must work at the same scope at which you define the SSL configuration and not at a higher scope. For example, if you create a key store that contains a certificate whose host name is part of the distinguished name (DN), then store that keystore in the node directory of the configuration repository. If you decide to create a certificate for the application server, then store that keystore on the application server in the application server directory.

When you configure the trust stores, which control trust decisions on the server, you must consider how much you want to isolate the application servers. You cannot isolate the application servers from the node agents or the deployment manager. However, you can configure the SOAP connector end points with the same personal certificate or to share trust. Naming persistence requires IIOP connections when they pass through the deployment manager. Because application servers always connect to the node agents when the server starts, the IIOP protocol requires that WebSphere Application Server establish trust between the application servers and the node agents.

Establishing node SSL isolation

By default, WebSphere Application Server installation uses a single chained certificate for each node so you can isolate nodes easily. A common trust store, which is located in the cell directory of the configuration repository, contains all of the signers for each node that is federated into the cell. After federation, each cell process trusts all of the other cell processes because every SSL configuration references the common trust store.

You can modify the default configuration so that each node has its own trust store, and every application server on the node trusts only the node agent that uses the same personal certificate. You must also add the signer to the node trust store so that WebSphere Application Server can establish trust with the deployment manager. To isolate the node, ensure that the following conditions are met:

- The deployment manager must initiate connections to any process
- The node agent must initiate connections to the deployment manager and its own application servers

- The application servers must initiate connections to the applications servers on the same node, to its own node agent, and the deployment manager

Figure 1 shows Node Agent A contains a key.p12 keystore and a trust.p12 trust store at the node level of the configuration repository for node A.

Figure 36.

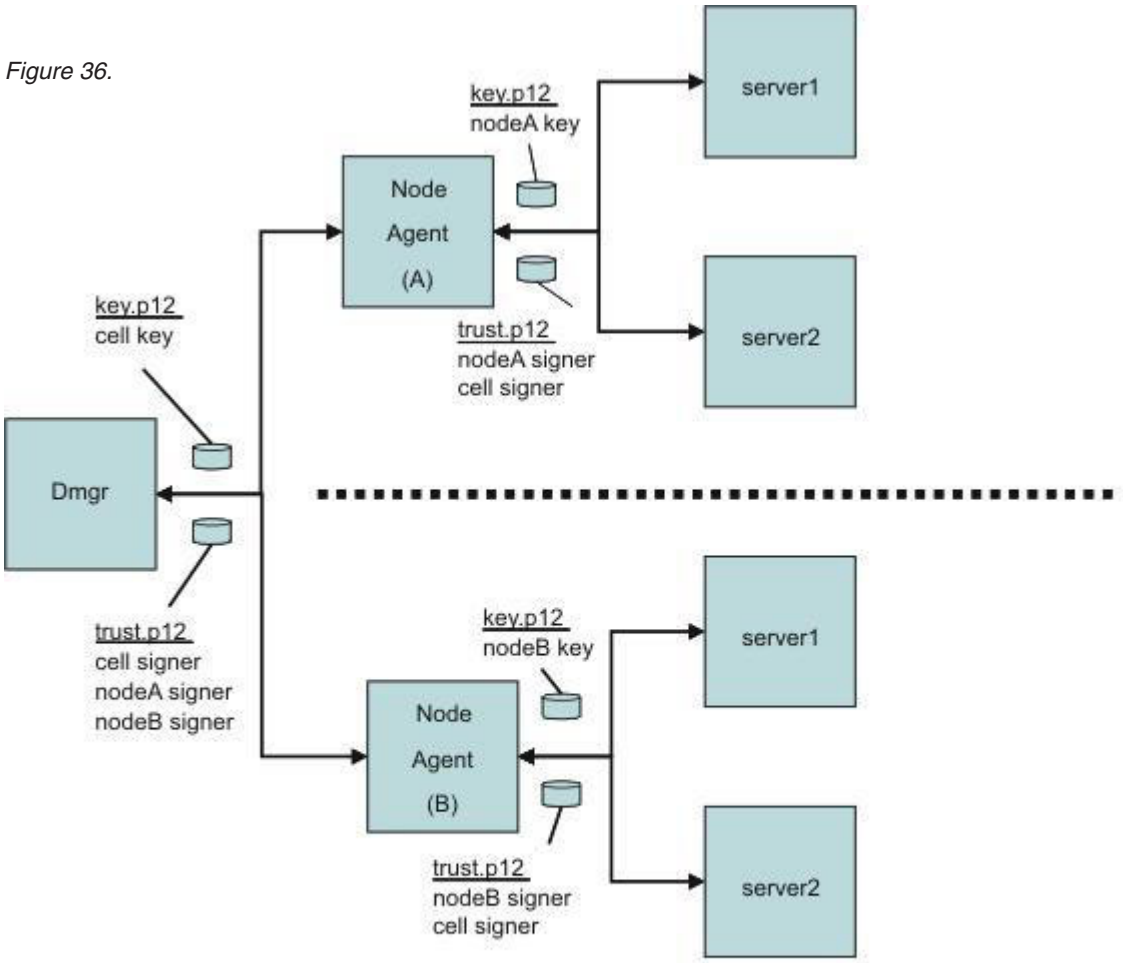


Figure 1: SSL Node Isolation

When you associate an SSL configuration with this keystore and truststore, you break the link with the cell-scoped trust store. To isolate the node completely, repeat this process for each node in the cell. WebSphere Application Server SSL configurations override the cell scope and use the node scope instead so that each process at this scope uses the SSL configuration and certificate alias that you selected at this scope. You establish proper administrative trust by ensuring that nodeA signer is in the common trust store and the cell signer is in the nodeA trust store. The same logic applies to node B as well. For more information, see Associating Secure Sockets Layer configurations centrally with inbound and outbound scopes.

Establishing application server SSL isolation

Isolating application server processes from one another is more challenging than isolating nodes. You must consider the following application design and topology conditions:

- An application server process might need to communicate with the node agent and deployment manager
- Isolating application server processes from each other might disable single sign-on capabilities for horizontal propagation

If you configure outbound SSL configurations dynamically, you can accommodate these conditions. When you define a specific outbound protocol, target host, and port for each different SSL configuration, you can override the scoped configuration.

Figure 2 shows how you might isolate an application server completely, although in practice this approach would be more complicated.

Figure 37.

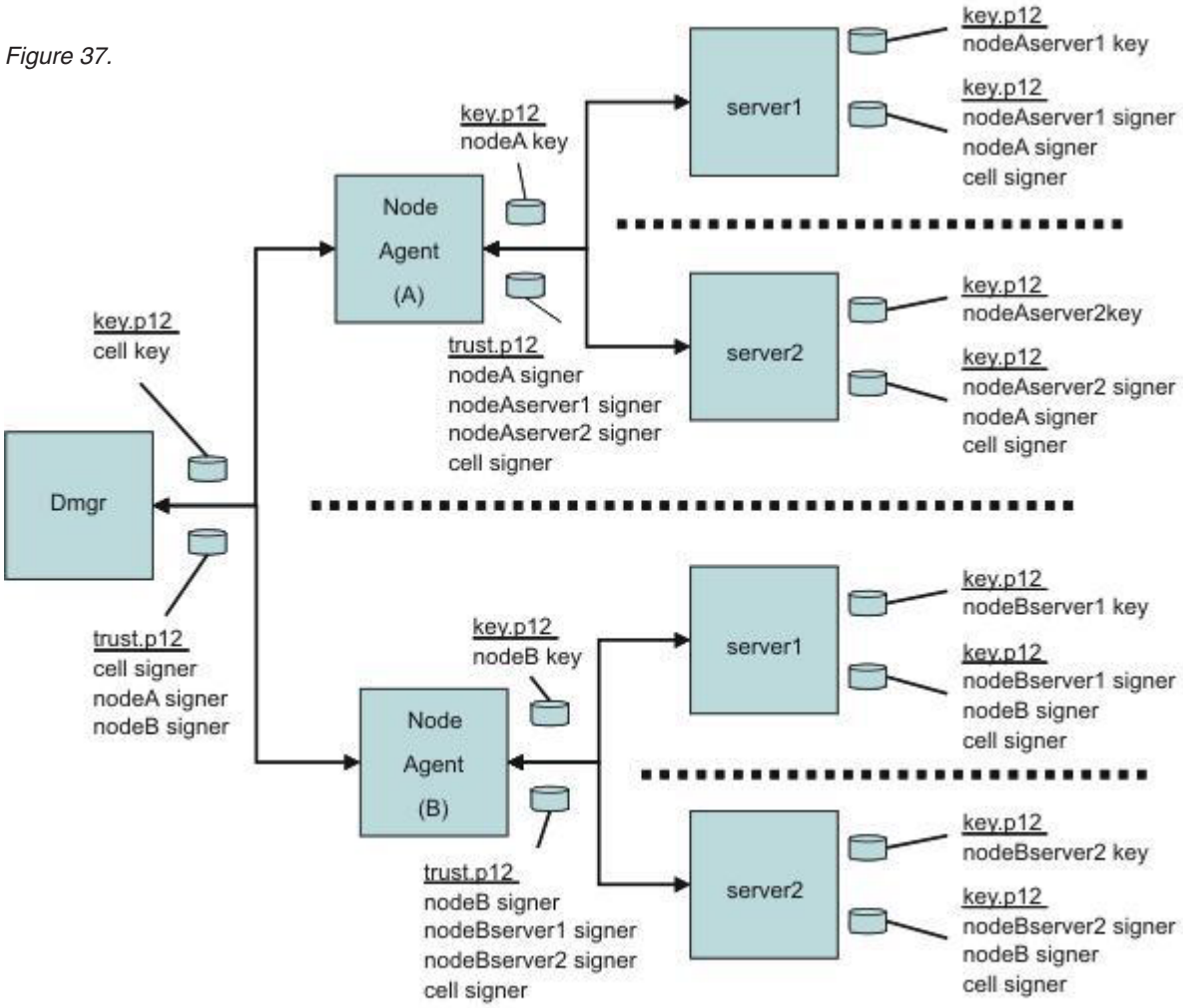


Figure 2: SSL Application Server Isolation

The dynamic configuration enables server1 on Node A to communicate with server 1 on Node B only over IIOP. The dynamic outbound rule is IIOP,nodeBhostname,*. For more information, see Associating a Secure Sockets Layer configuration dynamically with an outbound protocol and remote secure endpoint

Establishing cluster SSL isolation

You can configure application servers into clusters instead of scoping them centrally at the node or dynamically at the server to establish cluster SSL isolation. While clustered servers can communicate with

each other, application servers outside of the cluster cannot communicate with the cluster, thus isolating the clustered servers. For example, you might need to separate applications from different departments while maintaining a basic level of trust among the clustered servers. Using the dynamic outbound SSL configuration method previously described for servers, you can easily extend the isolated cluster as needed.

Figure 3 shows a sample cluster configuration where cluster 1 contains a key.p12 with its own self-signed certificate, and a trust.p12 that is located in the config/cells/<cellname>/clusters/<clustername> directory.

Figure 38.

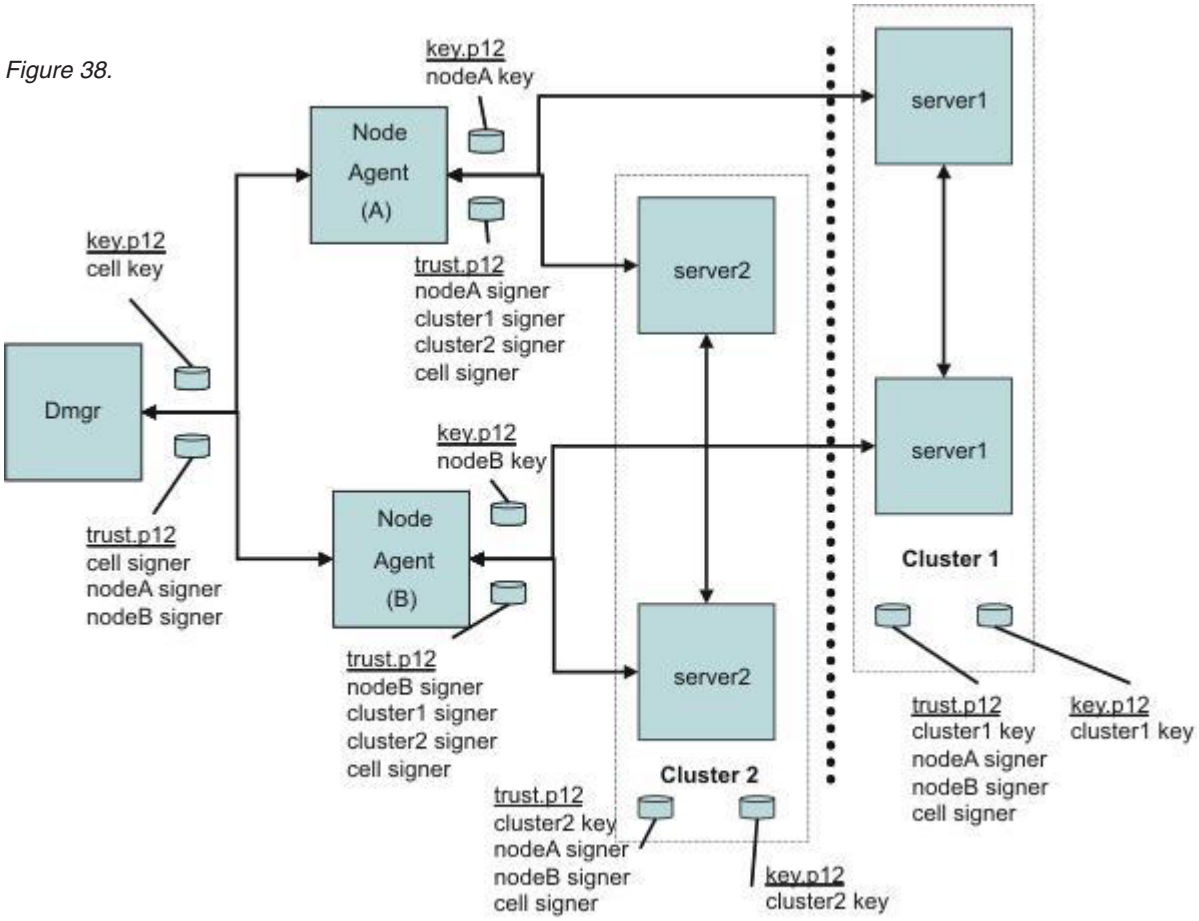


Figure 3: SSL Cluster Isolation

In the example, cluster1 might contain web applications, and cluster2 might contain EJB applications. Considering the various protocols, you decide to enable IIOP traffic between the two clusters. Your task is to define a dynamic outbound SSL configuration at the cluster1 scope with the following properties:

```
IIOP,nodeAhostname,9403|IIOP,nodeAhostname,9404|IIOP,nodeBhostname,9403|IIOP,nodeBhostname,9404
```

You must create another SSL configuration at the cluster1 scope that contains a new trust.p12 file with the cluster2 signer. Consequently, outbound IIOP requests go either to nodeAhostname ports 9403 and 9404 or to nodeBhostname ports 9403 and 9404. The IIOP SSL port numbers on these two application server processes in cluster2 identify the ports.

As you review Figure 3, notice the following features of the cluster isolation configuration:

- The trust.p12 for cluster1 contains signers that allow communications with itself (cluster1 signer), between both node agents (nodeAsigner and nodeBsigner), and with the deployment manager (cell signer).

- The trust.p12 for cluster2 contains signers that allow communications with itself (cluster2 signer), between both node agents (nodeAsigner and nodeBsigner), and with the deployment manager (cell signer).
- Node agent A and Node agent B can communicate with themselves, the deployment manager, and both clusters.

For more information, see Associating a Secure Sockets Layer configuration dynamically with an outbound protocol and remote secure endpoint .

Although this article presents an overview of isolation methods from an SSL perspective, you must also ensure that non-SSL ports are closed or applications require the confidentiality constraint in the deployment descriptor. For example, you can set the CSIv2 inbound transport panel to require SSL and disable the channel ports that are not secure from the server ports configuration.

Also, you must enable SSL client authentication for SSL to enforce the isolation requirements on both sides of a connection. Without mutual SSL client authentication, a client can easily obtain a signer for the server programmatically and thus bypass the goal of isolation. With SSL client authentication, the server would require the client's signer for the connection to succeed. For HTTP/S protocol, the client is typically a browser, a web service, or a URL connection. For the IIOP/S protocol, the client is typically another application server or a Java client. WebSphere Application Server must know the clients to determine if SSL client authentication enablement is possible. Any applications that are available through a public protocol must not enable SSL client authentication because the client may fail to obtain a certificate to authenticate to the server.

Note: It is beyond the scope of this article to describe all of the factors you must consider to achieve complete isolation.

Certificate options during profile creation

Starting in WebSphere Application Server Version 7.0, you have several options available during profile creation concerning the default certificate and root certificate of the server.

The new certificate options enable you to:

- Import the default certificate of the server
- Import the root certificate of the server
- Customize the default certificate subjectDN and validity period of the server
- Customize the root certificate subjectDN and validity period of the server

Two new panels are available during profile creation that enable you to make decisions about the default certificate and root certificate of the server.

The first panel, titled Security Certificate (Part 1), enables you to choose to import a certificate or to have WebSphere Application Server create the default certificate or the default root certificate of the server for you.

The second panel, titled Security Certificate (Part 2), either displays the information from the certificate imported from the previous panel, or, if you choose to have WebSphere Application Server create the certificate, enables you to change the subjectDN and the certificate validity period.

Customization of certificates can also be performed by using the manageprofile command and from a silent install response file.

Importing the default certificate of the server during profile creation

If the default certificate of the server is imported during profile creation, it is added to NodeDefaultKeyStore if on a stand-alone application server, or to CellDefaultKeyStore if on a deployment manager. The imported certificate signer is added to NodeDefaultTrustStore or CellDefaultTrustStore.

To import the default certificate of the server, you must have a personal certificate stored and a keystore that you have access to. You must know the location, type and password of the keystore. On the Security Certificate (Part 1) panel, do the following:

1. Select **Import an existing default personal certificate**.
2. Type or select the keystore file name.
3. Enter the password of the keystore.
4. Select a keystore type from the pull-down list.
5. If you have correctly filled in all information from the previous 3 steps, you are able to select a certificate alias from the pull-down list.

The certificate you choose is imported to the default keystore of the server. The next panel, Security Certificate (Part 2) displays the issuedTo and issuedBy certificate information.

If you use the manageprofiles command to import the default certificate, the options are:

-importPersonalCertKS keystore_path
the keystore file location

-importPersonalCertKSType keystore_type
the type of the keystore

-importPersonalCertKSPassword keystore_password
the password to open the keystore

-importPersonalCertKSAlias keystore_alias
the alias of the certificate used from the keystore

Importing the root certificate of the server during profile creation

If the server root certificate is imported during profile creation, the certificate is added to NodeDefaultRootStore on a stand-alone application server or to DmgrDefaultRootStore on a deployment manager. The signer is pulled from the imported root certificate and added to NodeDefaultTrustStore or CellDefaultTrustStore. The root certificate is used by WebSphere Application Server to sign any chained certificates it creates. If no default certificate is provided during profile creation, WebSphere Application Server uses the root certificate to sign the default certificate of the server.

To import the default certificate of the server, you must have a personal certificate stored and a keystore that you have access to. You must know the location, type and password of the keystore. On the Security Certificate (Part 1) panel, do the following:

1. Select **Import an existing root signing certificate**.
2. Type or select the keystore file name.
3. Enter the password of the keystore.
4. Select a keystore type from the pull-down list.
5. If you have correctly filled in all information from the previous 3 steps, you are able to select a certificate alias from the pull-down list.

The certificate you choose is imported to the root keystore of the server. The next panel, Security Certificate (Part 2) displays the issuedTo and issuedBy certificate information.

If you use the manageprofiles command to import the root certificate, the options are:

-importSigningICertKS keystore_path
the keystore file location

-importSigningCertKSType keystore_type
the type of the keystore

-importSigningCertKSPassword keystore_password
the password to open the keystore

-importSigningCertKSAlias keystore_alias
the alias of the certificate used from the keystore

Customizing the default certificate created by WebSphere Application Server

If you choose to let WebSphere Application Server create the default certificate of the server, you can customize the subject distinguished name (DN) and the life span of the certificate.

To customize the default certificate of the server on the Security Certificate (Part 1) panel, do the following:

1. Select **Create a new default personal certificate**.
2. On the next panel, Security Certificate (Part 2), the Issued to distinguished name field contains the WebSphere Application Server default DN. Replace this with your customized DN.
3. In Expiration period in years, select the number of years you want the certificate to be valid for.

If you use the manageprofiles command to customize the default certificate, the options are:

-personalCertDN distinguished_name
the DN to give to the certificate

-personalCertValidityPeriod validity_period
the life span to give to the certificate

Customizing the root certificate created by WebSphere Application Server

If you choose to let WebSphere Application Server create the root certificate, you can customize the DN of the certificate and the life span of the certificate.

To customize the root certificate of the server on the Security Certificate (Part 1) panel, do the following:

1. Select **Create a new root signing certificate**.
2. On the next panel, Security Certificate (Part 2), the Issued by distinguished name field contains the WebSphere Application Server default root certificate DN. Replace this with your customized DN.
3. In Expiration period in years, select the number of years you want the root certificate to be valid for.

If you use the manageprofiles command to customize the root certificate, the options are:

-signingCertDN distinguished_name
the DN to give to the root certificate

-signingCertValidityPeriod validity_period
the life span to give to the root certificate

Default chained certificate configuration in SSL

When a WebSphere Application Server process starts for the first time, the Secure Sockets Layer (SSL) runtime initializes the default keystores and truststores that are specified in the SSL configuration.

The chained certificates created during profile creation have a 1 year life span by default. The default root certificate used to signer the default chained certificate has a life span of 15 years. The life span of the

default and the root certificates can be customized during profile creation. An advantage in this type of chained certificate is that only the signer from the root certificate is needed to establish trust. When the chained certificate is regenerated with the same root certificate, clients using that root signer certificate for trust do not lose their trust.

Default keystore and truststore properties

WebSphere Application Server creates the `key.p12` default keystore file and the `trust.p12` default truststore file during profile creation. A default, chained certificate is also created in the **key.p12** file. The root signer, or public key, of the chained certificate is extracted from the **key.p12** file and added to the **trust.p12** file. If the files do not exist during process startup, they are recreated during startup.

You can identify keystore and truststore defaults because of their suffixes: **DefaultKeyStore** and **DefaultTrustStore**. Also, in the SSL configuration, you must set the **fileBased** attribute to `true` so that the runtime only uses the default keystores and truststore.

On a base application server, default key and truststores are stored in the node directory of the configuration repository. For example, the default `key.p12` and `trust.p12` stores are created with the AppSrv01 profile name, the myhostNode01Cell name, and the myhostNode01 node name. The keystore and truststore are located in the following directories:

- C:\WebSphere\AppServer\profiles\AppSrv01\config\cells\myhostNode01Cell \nodes\myhostNode01\key.p12
- C:\WebSphere\AppServer\profiles\AppSrv01\config\cells\myhostNode01Cell \nodes\myhostNode01\trust.p12

The default password is **WebAS** for all default keystores generated by WebSphere Application Server. Change the default password after the initial configuration for a more secure environment.

Default chained certificate

The default chained certificate of the server along with a root self-signed certificate used to sign the default chained certificate are created during profile creation.

You can recreate the certificates with different information simply by deleting the `*.p12` files in `/config` and `/etc`. Change the four properties in the next code example to the values you want the certificates to contain, then restart the processes. This causes the server certificate in `/config` and the client certificate in `/etc` to differ.

The certificate properties in the next code example exist in the **ssl.client.props** file, but do not exist in the server configuration. However, you can use these values in the server configuration by adding them as custom security properties in the administrative console. Click **Security** > **Global security** > **Custom properties** to change the following properties:

```
com.ibm.ssl.defaultCertReqAlias=default_alias
com.ibm.ssl.defaultCertReqSubjectDN=cn=${hostname},ou=myhostNode01,ou=myhostNode01Cell,o=IBM,c=US
com.ibm.ssl.defaultCertReqDays=365
com.ibm.ssl.defaultCertReqKeySize=1024
com.ibm.ssl.rootCertSubjectDN=cn=${hostname},ou=Root Certificate, ou=myhostNode01,
ou=myhostNode01Cell,o=IBM,c=US
com.ibm.ssl.rootCertValidDays=7300
com.ibm.ssl.rootCertAlias=root
com.ibm.ssl.rootCertKeySize=1024
```

After changing the properties, complete the following actions:

1. Delete the default key.p12 keystore and trust.p12 truststore files for the application server, which contain the default chained certificate. If the keystore and truststore file do not exist, WebSphere Application Server automatically generates them and creates new default certificates using the previously listed property values.
2. Delete the root keystore, which is the root-key.p12 file, to regenerate the root certificate with the previously listed properties.
3. Restart the application server.

If a default_alias value already exists, the runtime appends _#, where the number sign (#) is a number that increases until it is unique in the keystore. ${hostname} is a variable that is resolved to the host name where it was originally created. The default expiration date of chained certificates is one year from their creation date.

The runtime monitors the expiration dates of chained certificates using the certificate expiration monitor. These chained certificates are automatically replaced along with any signer certificates when they are within the expiration threshold, which is typically 30 days before expiration. You can increase the default key size beyond 1024 bits only when the Java runtime environment policy files are unrestricted (that is, not exported). For more information, see “Certificate expiration monitoring in SSL” on page 702.

Default keystore and truststore configurations for new Base Application Server processes

The following sample code shows the default SSL configuration for a base application server. References to the default keystores and truststores files are highlighted.

```
<repertoire xmi:id="SSLConfig_1" alias="NodeDefaultSSLSettings"
managementScope="ManagementScope_1">
<setting xmi:id="SecureSocketLayer_1" clientAuthentication="false"
securityLevel="HIGH" enabledCiphers="" jsseProvider="IBMJSSE2" sslProtocol="SSL_TLS"
keyStore="KeyStore_1" trustStore="KeyStore_2" trustManager="TrustManager_1"
keyManager="KeyManager_1"/>
</repertoire>
```

Default keystore

In the following sample code, the keystore object that represents the default keystore is similar to the XML object.

```
<keyStores xmi:id="KeyStore_1" name="NodeDefaultKeyStore"
password="{xor}349dkckdd=" provider="IBMJCE" location="${WAS_INSTALL_ROOT}/config
/cells/myhostNode01Cell/nodes/myhostNode01/key.p12" type="PKCS12" fileBased="true"
hostList="" initializeAtStartup="true" managementScope="ManagementScope_1"/>
```

The **NodeDefaultKeyStore** keystore contains the personal certificate that represents the identity of the secure endpoint. Any keystore reference can use the ${WAS_INSTALL_ROOT} variable, which is expanded by the runtime. The PKCS12 default keystore type is in the most interoperable format, which means that it can be imported into most browsers. The myhostNode01Cell password is encoded. The management scope determines which server runtime loads the keystore configuration into memory, as shown in the following code sample:

```
<managementScopes xmi:id="ManagementScope_1" scopeName="
(cell):myhostNode01Cell:(node):myhostNode01" scopeType="node"/>
```

Any configuration objects that are stored in the **security.xml** file whose management scopes are outside the current process scope are not loaded in the current process. Instead, the management scope is loaded by servers that are contained within the myhostNode01 node. Any application server that is on the specific node can view the keystore configuration.

When you list the contents of the **key.p12** file to show the chained certificate, note that the common name (CN) of the distinguished name (DN) is the host name of the resident machine. This listing enables you to verify the host name by its URL connections. Additionally, you can verify the host name from a custom trust manager. For more information, see “Trust manager control of X.509 certificate trust decisions” on page 677.

Contents of default keystore

The following sample code shows the contents of the default **key.p12** file in a keytool list:

```
keytool -list -v -keystore c:\WebSphere\AppServer\profile\AppSrv01\profiles\config
\cells\myhostNode01Cell\nodes\myhostNode01\key.p12 -storetype PKCS12 -storepass *****

Keystore type: PKCS12
Keystore provider: IBMJCE

Your keystore contains 1 entry

Alias name: default
Creation date: Dec 31, 1969
Entry type: keyEntry
Certificate chain length: 2
Certificate[1]:
Owner: CN=myhost.austin.ibm.com, OU=myhostNode01Cell, OU=myhostNode01, O=IBM, C=US
Issuer: CN=myhost.austin.ibm.com, OU=Root Certificate, OU=myhostNode01Cell, OU=myhostNode01, O=IBM, C=US
```

```
Serial number: 4e48f29aafea6
Valid from: 2/7/08 1:03 PM until: 2/6/09 1:03 PM
Certificate fingerprints:
  MD5:  DB:FE:65:DB:40:13:F4:48:A4:CE:2F:4F:60:A5:FF:2C
  SHA1: A1:D4:DD:4B:DE:7B:45:F7:4D:AA:6A:FC:92:38:78:53:7A:99:F1:DC
Certificate[2]:
Owner: CN=myhost.austin.ibm.com, OU=Root Certificate, OU=myhostNode01Cell, OU=myhostNode01, O=IBM, C=US
Issuer: CN=myhost.austin.ibm.com, OU=Root Certificate, OU=myhostNode01Cell, OU=myhostNode01, O=IBM, C=US
Serial number: 4e48e5fd4eae3
Valid from: 2/7/08 1:03 PM until: 2/2/28 1:03 PM
Certificate fingerprints:
  MD5:  A5:9B:05:78:CF:AB:89:94:C9:2E:F1:87:34:B3:FC:75
  SHA1: 43:74:B6:C7:FA:C1:0F:19:F2:51:2B:17:60:0D:34:93:55:BF:D5:D2

*******************************************
*******************************************
```

The default alias name and the keyEntry entry type indicate that the private key is stored with the public key, which represents a complete personal certificate. The certificate is owned by`CN=myhost.austin.ibm.com, OU=myhostNode01Cell, OU=myhostNode01, O=IBM, C=US` and it is issued by the default root certificate, which is owned by`CN=myhost.austin.ibm.com, OU=Root Certificate, OU=myhostNode01Cell, OU=myhostNode01, O=IBM, C=US` By default, the certificate is valid for one year from the date of creation.

Additionally, in some signer-exchange situations, the certificate fingerprint ensures that the sent certificate has not been modified. The fingerprint, which is a hash algorithm output for the certificate, is displayed by the WebSphere Application Server runtime during an automated signer exchange on the client side. The client fingerprint must match the fingerprint that is displayed on the server. The runtime typically uses the SHA1 hash algorithm to generate certificate fingerprints.

Default truststore

In the following sample code, the keystore object represents the default `trust.p12` truststore. The truststore contains signer certificates that are necessary for making trust decisions:

```
<keyStores xmi:id="KeyStore_2" name="NodeDefaultTrustStore"
password="{xor}349dkckdd=" provider="IBMJCE" location="${WAS_INSTALL_ROOT}
/config/cells/myhostNode01Cell/nodes/myhostNode01/trust.p12" type="PKCS12"
fileBased="true" hostList="" initializeAtStartup="true" managementScope="ManagementScope_1"/>
```

Contents of default truststore

The following sample code shows the contents of the default `trust.p12` truststore in a keytool listing. By default, for the sample chained certificate, the root certificate signer is included in the trust store. The root signer alias name and the trustedCertEntry entry type indicate that the certificate is the public key. The private key is not stored in this truststore. In addition, all truststores contain the default DataPower certificate.

```
keytool -list -v -keystore c:\WebSphere\AppServer\profile\AppSrv01\profiles\config\cells\myhostNode01Cell
\nodes\myhostNode01\trust.p12 -storetype PKCS12 -storepass *****
```

```
Keystore type: PKCS12
Keystore provider: IBMJCE

Your keystore contains 2 entries

Alias name: root
Creation date: Dec 31, 1969
Entry type: trustedCertEntry

Owner: CN=myhost.austin.ibm.com, OU=Root Certificate, OU=myhostNode01Cell, OU=myhostNode01, O=IBM, C=US
Issuer: CN=myhost.austin.ibm.com, OU=Root Certificate, OU=myhostNode01Cell, OU=myhostNode01, O=IBM, C=US
Serial number: 4e48e5fd4eae3
Valid from: 2/7/08 1:03 PM until: 2/2/28 1:03 PM
Certificate fingerprints:
   MD5:  A5:9B:05:78:CF:AB:89:94:C9:2E:F1:87:34:B3:FC:75
   SHA1: 43:74:B6:C7:FA:C1:0F:19:F2:51:2B:17:60:0D:34:93:55:BF:D5:D2

*******************************************
*******************************************

Alias name: datapower
Creation date: Dec 31, 1969
Entry type: trustedCertEntry

Owner: OU=Root CA, O="DataPower Technology, Inc.", C=US
Issuer: OU=Root CA, O="DataPower Technology, Inc.", C=US
Serial number: 0
```

```
Valid from: 6/11/03 1:23 PM until: 6/6/23 1:23 PM
Certificate fingerprints:
   MD5:  18:AC:86:D1:9A:90:A2:AE:8B:28:F9:A8:75:C8:A9:DB
   SHA1: A9:BA:A4:B5:BC:26:2F:5D:2A:80:93:CA:BA:F4:31:05:F2:54:14:17
```

Secure installation for client signer retrieval in SSL

Each profile in the WebSphere Application Server environment contains a unique chained certificate signed by a unique long lived root certificate that was created when the profile was created. This certificate replaces the default self-signed certificate that ships with WebSphere Application Server Version 6.1 as well as the default dummy certificate that ships in releases prior to Version 6.1. When a profile is federated to a deployment manager, the signer for the root signing certificate is added to the common truststore for the cell, establishing trust for all certificates signed by that root certificate.

Note: Do not use the dummy keystore and truststore files, which are referenced in this topic, in a production environment. These files contain the same certificates and are used everywhere, which is not secure. Also, change the passwords for the keystore and truststore so that it does not use the `WebAS` default password.

By default, clients do not trust servers from different profiles in the WebSphere Application Server environment. That is, they do not contain the root signer for these servers. There are some things that you can do to assist in establishing this trust:

1. Enable the signer exchange prompt to accept the signer during the connection attempt.
2. Run the **retrieveSigners** utility to download the signers from that system prior to making the connection.
3. Copy the `trust.p12` file from the `/config/cells/<cell_name>/nodes/<node_name>` directory of the server profile to the `/etc` directory of the client. Update the SSL configuration to reflect the new file name and password, if they are different. Copying the file provides the client with a `trust.p12` that contains all signers from servers in that cell. Also, you might need to perform this step for back-level clients that are still using the `DummyClientTrustFile.jks` file. In this case, you might need to change the `sas.client.props` or `soap.client.props` file to reflect the new truststore, truststore password, and truststore type (**PKCS12**).

For clients to perform an in-band signer exchange, you must specify the `ssl.client.props` file as a com.ibm.SSL.ConfigURL property in the SSL configuration. For managed clients, this is done automatically. Signers are designated either as in-band during the connection or out-of-band during runtime. You must also set the com.ibm.ssl.enableSignerExchangePrompt attribute to `true`.

Tip: You can configure a certificate expiration monitor to replace server certificates that are about to expire. For more information about how clients can retrieve the new signer from the configuration, see “Certificate expiration monitoring in SSL” on page 702.

Using the signer exchange prompt to retrieve signers from a client

When the client does not already have a signer to connect to a process, you can enable the signer exchange prompt. The signer exchange prompt displays once for each unique certificate and for each node. After the signer for the node is added, the signer remains in the client truststore. The following sample code shows the signer exchange prompt retrieving a signer from a client:

```
C:\WASX_e0540.11\AppServer\profiles\AppSrv01\bin\serverStatus -all ADMU0116I: Tool information is being logged in file
C:\WASX_e0540.11\AppServer\profiles\AppSrv01\logs\serverStatus.log ADMU0128I: Starting tool with the
AppSrv01 profile ADMU0503I:
Retrieving server status for all servers ADMU0505I: Servers found in configuration:
ADMU0506I: Server name: dmgr
*** SSL SIGNER EXCHANGE PROMPT *** SSL signer from target host 192.168.1.5 is not found in truststore
C:\WebSphere\AppServer\profiles\AppSrv01\etc\trust.p12.

Here is the signer information (verify the digest value matches what is  displayed at the server):
Subject DN:    CN=myhost.austin.ibm.com, OU=myhostNode01Cell, OU=myhostNode01, O=IBM, C=US
Issuer DN:     CN=myhost.austin.ibm.com, OU=Root Certificate, OU=myhostNode01Cell, OU=myhostNode01,
```

```
O=IBM, C=US
Serial number: 2510775664686266 Expires:       Thu Feb 19 15:58:49 CST 2009
SHA-1 Digest:  2F:96:70:23:08:58:6F:66:CD:72:61:E3:46:8B:39:D4:AF:62:98:C3
MD5 Digest:    04:53:F8:20:A2:8A:6D:31:D0:1D:18:90:3D:58:B9:9D

Subject DN: CN=myhost.austin.ibm.com, OU=Root Certificate, OU=myhostNode01Cell,
OU=myhostNode01, O=IBM, C=US Issuer DN:  CN=myhost.austin.ibm.com, OU=Root
Certificate, OU=myhostNode01Cell, OU=myhostNode01, O=IBM, C=US Serial number:
2510773295548841 Expires: Tue Feb 15 15:58:46 CST 2028 SHA-1 Digest:
2F:96:70:23:08:58:6F:66:CD:72:61:E3:46:8B:39:D4:AF:62:98:C3
MD5 Digest: 04:53:F8:20:A2:8A:6D:31:D0:1D:18:90:3D:58:B9:9D
Add signer to the truststore now? (y/n) y A retry of the request may need to occur. ADMU0508I:
The Deployment Manager "dmgr" is STARTED
```

To automate this process, see retrieveSigners command.

When a prompt occurs to accept the signer, a socket timeout can occur and the connection might be broken. For this reason, the message A retry of the request may need to occur. displays after answering the prompt. The message informs the user to resubmit the request. This problem should not happen frequently, and it might be more prevalent for some protocols than others.

A retry of the request may need to occur if the socket times out while waiting for a prompt response. If the retry is required, note that the prompt will not be re-displayed if (`y`) is entered, which indicates the signer has already been added to the trust store.

Verify the displayed SHA-1 digest, which is the signature of the certificate that is sent by the server. If you look at the certificate on the server, verify that the same SHA-1 digest displays.

You can disable the prompt when you do not want it to display by running the **retrieveSigners** utility to retrieve all of the signers for a particular cell. You can download or upload the signers from any remote keystore to any local keystore by referencing a common truststore with this client script. For more information, see Default chained certificate configuration in SSL.

Using the retrieveSigners utility to download signers for a client

You can run the **retrieveSigners** utility to retrieve all of the signers from the remote keystore for a specified client keystore.

The typical remote keystore to reference is NodeDefaultTrustStore.

The truststore contains the signers that enable the client to connect to its processes. The **retrieveSigners** utility can point to any keystore in the target configuration, within the scope of the target process, and can download the signers (certificate entries only) to any client keystore in the `ssl.client.props` file.

Obtaining signers for clients and servers from a previous release

Note: When a client from a release prior to version 7.0 connects to the current release, the client must obtain signers for a successful handshake. Clients using previous releases of WebSphere Application Server cannot obtain signers as easily as in the current release. You can copy the deployment manager common truststore to your back-level client or server, and then re-configure the SSL configuration to directly reference that truststore. This common truststore of type `PKCS12` is located in the `/config/cells/<cell_name>/nodes/<node_name>` directory in the configuration repository and has a default password of `WebAS`.

To collect all of the signers for the cell in a single `trust.p12` keystore file, complete following steps:

1. Copy the trust.p12 keystore file on the server and replicate it on the client. The client references the file directly from the `sas.client.props` and `soap.client.props` files that specify the SSL properties for previous releases.

2. Change the client-side keystore password so that it matches the default cell name that is associated with the copied keystore.
3. Change the default keystore type for the `trust.p12` file to **PKCS12** in the client configuration.

The following two code samples show you a before and an after view of the changes to make.

Default SSL configuration of `sas.client.props` for a previous release

```
com.ibm.ssl.protocol=SSL com.ibm.ssl.keyStore=file\:///  C\:/SERV1_601_0208/AppServer/profiles/AppSrv01/etc/
DummyClientKeyFile.jks com.ibm.ssl.keyStorePassword={xor}CDo9Hgw\= com.ibm.ssl.keyStoreType=JKS
com.ibm.ssl.trustStore=
file\:/// C\:/SERV1_601_0208/AppServer/profiles/AppSrv01/etc/DummyClientTrustFile.jks
com.ibm.ssl.trustStorePassword={xor}CDo9Hgw\=
com.ibm.ssl.trustStoreType=JKS
```

SSL configuration changes that are required to common truststore file in the `/etc` directory of the client

```
com.ibm.ssl.protocol=SSL com.ibm.ssl.keyStore=file\:/// C\:/SERV1_601_0208/AppServer/profiles/AppSrv01/etc/
DummyClientKeyFile.jks com.ibm.ssl.keyStorePassword={xor}CDo9Hgw\= com.ibm.ssl.keyStoreType=JKS
com.ibm.ssl.trustStore=file\:/// C\:/SERV1_601_0208/AppServer/profiles/AppSrv01/etc/trust.p12
com.ibm.ssl.trustStorePassword=myhostNode01Cell com.ibm.ssl.trustStoreType=PKCS12
```

Tip: You can run the **PropsFilePasswordEncoder** script, which is located in the `/bin` directory to encode the password.

You can also make these changes in the `soap.client.props` file and specify the `key.p12` file in place of the `DummyClientKeyFile.jks` file. However, you must also change the keyStorePassword and keyStoreType values to match those in the default `key.p12` file.

In releases of WebSphere Application Server prior to Version 7.0, you must edit the SSL configuration on the server to replace the common truststore. The `trust.p12` file, which is used by the server, also must contain the default dummy certificate signer for connections among servers at previous release levels. You might need to manually extract the default certificate from the `DummyServerKeyFile.jks` file and then import the certificate into the `trust.p12` file that you added to the configuration.

retrieveSigners command:

The retrieveSigners command creates a new client self-signed certificate, keystore, and SSL configuration in the ssl.client.props file. Using this command you can optionally extract the signer to a file.

For more information about where to run this command, read about Using command tools.

Syntax

Use the following command syntax to create a new client self-signed certificate, keystore, and SSL configuration in the ssl.client.props file.

```
retrieveSigners <remoteKeyStoreName> <localKeyStoreName> [options]
```

The <remoteKeyStoreName> and <localKeyStoreName> parameters are required. The following optional parameters are available:

```
[-remoteAlias aliasFromRemoteStore]
[-localAlias storeAsAlias]
[-listRemoteKeyStoreNames][-listLocalKeyStoreNames]
[-autoAcceptBootstrapSigner][-uploadSigners] [-host host]
[-port port][-conntype JSR160RMI|RMI|SOAP|IPC][-user user]
[-password password]
[-trace] [-logfile filename]
[-replacelog] [-quiet] [-help]
```

Parameters

The following parameters are available for the **retrieveSigners** command:

-remoteKeyStoreName
The name of a truststore that is located in the server configuration from which to retrieve the signers. This parameter is typically the CellDefaultTrustStore file for a managed environment or the NodeDefaultTrustStore file for an unmanaged environment.

-localKeyStoreName
The name of the truststore that is located in the ssl.client.props file for the profile to which the retrieved signers is added. This parameter is typically the ClientDefaultTrustStore file for either a managed or unmanaged environment.

-remoteAlias <aliasFromRemoteStore>
Specifies one alias from the remote truststore that you want to retrieve. Otherwise, all signers from the remote truststore are retrieved.

-localAlias <storeAsAlias>
Determines the name of the alias stored in the local truststore. This option is only valid if you specify the –remoteAlias option. If you do not specify the -localAlias option, then the alias name from the remote truststore is used, if possible. If an alias clash occurs, then the alias name is used and has an incremented number appended to the end of it until a unique alias is found.

-listRemoteKeyStoreNames
Sends a remote request to the server to list all keystores that you can specify for the remoteKeyStoreName parameter. Use this command when you are unsure of the name of the remote truststore from which you want to download the signers.

-listLocalKeyStoreNames
Lists the keystores located in the ssl.client.props file that you can specify for the localKeyStoreName parameter. This truststore receives the signers from the server. Use this parameter when you are unsure of the name of the local truststore into which you want to retrieve the signers. The default name of the truststore is ClientDefaultTrustStore and is located in the ssl.client.props file.

-autoAcceptBootstrapSigner
Automatically adds a signer to make a secure connection to the server. The purpose of the option is to support automation of the command so that you do not need to accept the signer. After the signer is added to the local truststore, an SHA hash prints so that you can verify the certificate.

-uploadSigners
Converts the signer download into a signer upload. The signers from the localKeyStoreName parameter is sent to the remoteKeyStoreName parameter instead.

-host <host>
Specifies the target host from which the signers are retrieved.

-port <port>
Specifies the target administrative port to which you want to connect. You must specify the port based on the -conntype parameter. If the conntype is SOAP, the default port is 8879. This value can vary for different servers. If the conntype is RMI, the default port is 2809.

-conntype <JSR160RMI|IPC|RMI|Soap>
Determines the administrative connector type that is used for the MBean call to retrieve the signers.

Note: Eventually switch from the RMI connector to the JSR160RMI connector because support for the RMI connector is deprecated.

-user <user>
When the -uploadSigners flag is used, you are required to specify this option to supply the user name that is authenticated for the MBean operation. If you do not specify this parameter when the -uploadSigners flag is used, then you are prompted for credentials by default.

-password <password>
When the -uploadSigners flag is used, you are required to specify this option to supply the password that is authenticated for the MBean operation. The password goes along with the –user parameter.

-trace
When specified, this parameter enables tracing of the trace specification necessary to debug this component. By default, the trace is located in the profiles/*profile_name*/log/retrieveSigners.log file.

-logfile <filename>
Overrides the default trace file. By default, the trace will appear in the profiles/*profile_name*/log/retrieveSigners.log file.

-replacelog
Causes the existing trace file to be replaced when the command runs.

-quiet
Suppresses most messages from printing to the console.

-help
Prints a usage statement.

-? Prints a usage statement.

Usage scenario

The following examples demonstrate correct syntax for using the **retrieveSigners** command:

- The following example lists remote and local keystores:

```
retrieveSigners.bat -listRemoteKeyStoreNames -listLocalKeyStoreNames -conntype RMI -port 2809 [Windows systems]

retrieveSigners.sh -listRemoteKeyStoreNames -listLocalKeyStoreNames -conntype RMI -port 2809 [Unix systems]
```

Example output

```
CWPKI0306I: The following remote keystores exist on the specified server:
          CMSKeyStore, NodeLTPAKeys, NodeDefaultTrustStore, NodeDefaultKeyStore
CWPKI0307I: The following local keystores exist on the client:
          ClientDefaultKeyStore, ClientDefaultTrustStore
```

- The following example retrieves all signers from NodeDefaultTrustStore:

```
retrieveSigners.bat NodeDefaultTrustStore ClientDefaultTrustStore -autoAcceptBootstrapSigner
-conntype RMI -port 2809 [Windows]

retrieveSigners.sh NodeDefaultTrustStore ClientDefaultTrustStore -autoAcceptBootstrapSigner
-conntype RMI -port 2809 [Unix]
```

Example output

```
CWPKI0308I: Adding signer alias "CN=BIRKT40.austin.ibm.com, O=IBM, C=US" to
          local keystore "ClientDefaultTrustStore" with the following SHA
          digest: 40:20:CF:BE:B4:B2:9C:F0:96:4D:EE:E5:14:92:9E:37:8D:51:A5:47
```

Certificate expiration monitoring in SSL

The certificate expiration monitor administrative task is a scheduled task that cycles through all the keystores in the security configuration and reports on any certificates that are expired, certificates that fall within the expiration threshold, and certificates that fall within the pre-notification period.

Note: This topic references one or more of the application server log files. As a recommended alternative, you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log` , `SystemErr.log`, `trace.log`, and `activity.log` files on distributed and IBM i systems. You can also use HPEL in conjunction with your native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

The certificate expiration monitor also replaces self-signed and chained certificates that have a root in the root keystore if configured to do so. If the monitor is configured to replace certificates, then certificates that

are expired or fall in the expiration threshold are replaced. Certificates that are imported from an external Certificate authority (CA) are reported but not replaced.

Certificate expiration monitoring relies on the following definitions:

Expired certificates
Certificates are created with a finite life span. Self-signed or chained certificates that have reached the end of their life span are reported and replaced, if possible. Certificate authority signed certificates cannot be replaced but will be reported. Replacing CA-signed certificates is the responsibility of the administrator.

Certificates within the expiration threshold
There is a period of time before a certificate expires. A certificate in this period of time is one within the expiration threshold. The server replaces certificates within the expiration threshold so that the certificate does not expire and cause outages. By default the expiration threshold is 60 days, but can be configured as required.

Pre-notification period
Before a certificate falls within the expiration threshold there are warnings issued that indicate that the certificate will be replaced, when the expiration threshold date is reached. The period of time prior to the expiration threshold date is called the pre-notification period and is set at 90 days for the certificate.

The certificate expiration monitor performs the following:

1. Clears out the **NodeDefaultDeletedStore** or **DmgrDefaultDeletedStore**. This operation is performed silently without reporting that the certificates are deleted.
2. Checks the root key stores, **DmgrDefaultRootStore** or **NodeDefaultRootStore** and the **DmgrRSATokenRootStore** or **NodeRSATokenRootStore**. If any root certificates are expired, falls in the threshold period, or the pre-notification period, then the certificate is noted in the report.
3. If there are any root certificates that are expired or fall in the threshold period that root certificate is recreated using all the information used to create the original one. Any signer certificates from the original root certificate are replaced with the signers from the new root certificate.
4. If a root certificate is replaced, then all the keystores are checked to see if there are any chained certificates signed with the original root certificate. If there are, then the chain certificate is renewed (recreated with the new root certificate). Any signer certificate from the original certificate is replaced with the signer from the recreated certificate.
5. After all root keystores are processed, the rest of the keystores are checked for expired certificates, certificates in the expiration threshold, or certificates in the pre-notification period. Any certificate falling in any one of these categories is noted in the report.
6. If there are any expired certificates or certificates in the expiration threshold period and these certificates are self-signed certificates or chained certificates created by WebSphere, then they are replaced. If the chained certificates root is not in the root key store then it will be recreated as a default root certificate. Any signer certificates from the original certificate are replaced with the signer from the new certificate."
7. A report is generated and returned, written to a log file, or mailed.

The server default certificate is a chained certificate with a 365 day life span. It is signed with the default root certificate which has a 15 year life span.

You can configure this monitor task to run according to a particular schedule. The schedule produces the next start date that persists in the configuration and, when the date is reached, WebSphere Application Server starts the monitor to check all of the keystores for certificates that meet the expiration threshold. You can start the task manually to run at any time.

The following security.xml configuration object specifies when the monitor task starts, determines the certificate expiration threshold, and indicates whether you are notified in an email using Simple Mail Transfer Protocol (SMTP) or in a message log.

```
<wsCertificateExpirationMonitor xmi:id="WSCertificateExpirationMonitor_1"
name="Certificate Expiration Monitor" daysBeforeNotification="30"
isEnabled="true" autoReplace="true" deleteOld="true"
wsNotification="WSNotification_1" wsSchedule="WSSchedule_2"
nextStartDate="1134358204849"/>
```

The expiration monitor replaces self-signed certificates and chained personal certificates that are signed by a root certificate in **DmgrDefaultRootStore** or **NodeDefaultRootStore**. Self-signed certificates are renewed using all the information that was used to create the original self-signed certificate. A chained certificate is renewed using the same root certificate that was used to sign the original certificate.

The expiration monitor automatically replaces only self-signed certificates and chained certificates that are expired or that meet the expiration threshold criteria. To replace all of the signers from the old certificate with the signer that belongs to the new certificate in all the keystores in the configuration for that cell, set the autoReplace attribute to true. When the deleteOld attribute is true, the old personal certificate and old signers also are deleted from the keystores. The isEnabled attribute determines whether the expiration monitor task runs based upon the nextStartDate attribute that is derived from the schedule. The nextStartDate attribute is derived from the schedule in milliseconds since 1970, and is identical to the System.currentTimeMillis(). If the nextStartDate has already passed when an expiration monitor process begins, and the expiration monitor is enabled, the task is started, but a new nextStartDate value is established based on the schedule.

The following sample the schedule object shows the frequency attribute as the number of days between each run of the certificate monitor.

```
<wsSchedules xmi:id="WSSchedule_2" name="ExpirationMonitorSchedule"
frequency="30" dayOfWeek="1" hour="21" minute="30"/>
```

The dayOfWeek attribute adjusts the schedule to run on a specified day of the week, which is always the same day regardless of whether the frequency is set to 30 or 31 days. Based on 24-hour clock, the hour and minute attributes determine when the expiration monitor is started on the specified day.

The following sample code of the notification object shows the notification configuration, which notifies you after the expiration monitor runs.

```
<>wsNotifications xmi:id="WSNotification_1" name="MessageLog" logToSystemOut="true" emailList=""/
```

For expiration monitor notifications, you can select message log, email using SMTP server, or both methods of notification. When you configure the email option, use the format user@domain@smtpserver. If you do not specify an SMTP server, WebSphere Application Server defaults to the same domain as the email address. For example, if you configure joeuser@ibm.com, WebSphere Application Server attempts to call smtp-server.ibm.com. To specify multiple email addresses using scripting, you must add a pipe (|) character between entries. When you specify the logToSystemOut attribute, the expiration monitor results are sent to the message log for the environment, which is typically the SystemOut.log file.

The expiration monitor clears out the deleted certificates keystore. The monitor first clears out the deleted keystore. Due to the nature of the PKCS12 keystore, there must be at lease one entry in the keystore so the signer certificates from the dummy key store will remain in the deleted keystore. There is no reporting on the certificate being deleted from the deleted keystore.

Important: When the expiration monitor replaces certificates, this can dynamically affect the runtime when the following configuration option is enabled:

Security > **SSL certificate and key management**. Under configuration settings, check the checkbox for **Dynamically update the run time when SSL configuration changes occur**.

When enabled, any certificates that are replaced causes the client SSL runtime to begin using the new certificates immediately, which in turn, flushes SSL and keystore caches and causes some ports using SSLServerSockets (RMI/IIOP on distributed and Admin SOAP) to restart. Restarting ports breaks existing connections. These connections can be reconnected after the port restart is completed. Endpoints using the channel framework (HTTP, BUS, RMI/IIOP on z/OS) leave existing connections unaffected but still use the new certificates for new connections.

When the dynamic change property is disabled and before the new certificates become effective, the administrator needs to recycle all processes in the entire cell after each node has the synchronized configuration. Regardless of which method is chosen, you should always check the health of your cell after the certificate expiration monitor has run (based on the schedule specified). The schedule should be set to run the certificate expiration monitor during a maintenance period so that if a restart is required after the certificate replacement, it will not cause unexpected outages.

Dynamic configuration updates in SSL

During the Secure Sockets Layer (SSL) runtime, dynamic configuration updates affect both inbound and outbound SSL endpoints. For inbound SSL endpoints, the changes that are implemented by the SSL channel are only affected by dynamic changes. For outbound SSL endpoints, all outbound connections inherit the new configuration changes.

In this release, dynamic update functionality provides you with greater flexibility and efficiency. You can change SSL configurations without restarting WebSphere Application Server for the changes to take effect.

To make dynamic changes, in the administrative console click **Security** > **SSL certificates and key management**, then select the **Dynamically update the runtime when SSL configuration changes occur** check box. You must save your changes and then synchronize the `security.xml` file with remote systems. A remote system must be able to confirm that **dynamicallyUpdateSSLConfig=true** is in the `security.xml` file.

The SSL runtime reloads the modified SSL configuration and creates a new SSLEngine for the modified connections that are associated with inbound endpoints. New outbound connections use the new configuration while existing connections continue to use the old SSLEngine object and are not affected.

Tip: Make dynamic changes to the SSL configuration during off-peak hours. Synchronization delays can negatively affect connections when you update SSL configurations during peak hours.

You can turn on and off the dynamicallyUpdateSSLConfig attribute in the `security.xml` file to ensure successful updates by doing the following actions:

1. Set dynamicallyUpdateSSLConfig=On.
2. Save the updated configuration.
3. Synchronize the `security.xml` file with remote systems.
4. Set the dynamicallyUpdateSSLConfig attribute to `Off`.

You must verify that all of the nodes receive the changes before turning off the dynamicallyUpdateSSLConfig attribute. Test the changes in a test environment before updating the production environment.

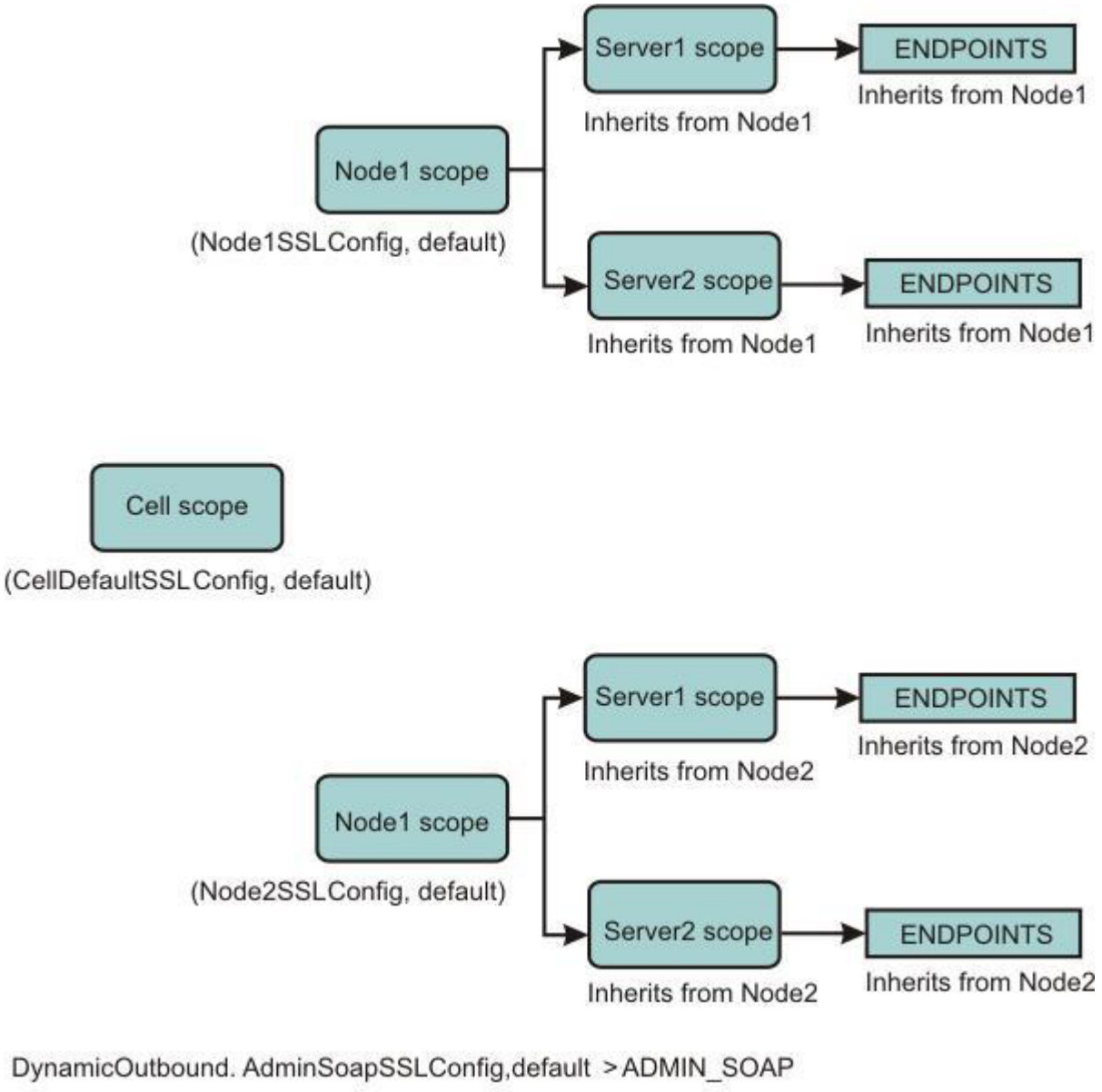
Tip: Some SSL changes, especially administrative SSL changes, can cause server outages if you fail to test them first. When a change prevents trust between two endpoints, the endpoints cannot communicate with each other. Additionally, if administrative SSL connection updates cause system outages, you might need to disable the nodes after you make corrective changes using the deployment manager. From the command line, you can manually synchronize the server to retrieve the new SSL changes, then restart the nodes.

Management scope configurations

Inbound and outbound management scopes represent opposing directions during the connection handshake process. To view inbound and outbound management scopes, use the topology tree view in the administrative console. You can define Secure Sockets Layer (SSL) configurations to distinguish the connection requirements for each direction inbound or outbound.

When expanded, the topology tree represents inbound and outbound connections for each management scope, cell, node group, node, server, cluster, and endpoint. Inbound endpoints require a server certificate. The SSL configuration specifies the server certificate for server authentication. Outbound endpoints require validated signers. Outbound endpoints connect to one or more target servers; inbound endpoints receive requests from one or more clients. The set of peer endpoints for outbound connections is typically a subset of the set of peer endpoints for inbound connections, which means you must define different requirements for inbound and outbound connections.

The following figure shows an example of two nodes: Node1 and Node2. These two nodes are isolated from one another because their SSL configurations, truststore files, and keystore files are different.



DynamicOutbound. AdminSoapSSLConfig,default >ADMIN_SOAP

In the example of two nodes, note that Node1 cannot communicate with Node2, but each of the two nodes must be able to communicate with the deployment manager and its administrative functions. With dynamic outbound selection, you can choose an SSL configuration and a certificate alias that reference a common truststore. When a process requires the ADMIN_SOAP protocol for an outbound connection, the server uses this single SSL configuration. Because all of the scopes under the cell level inherit this configuration, all outbound connections can communicate with the deployment manager. See additional information about dynamic outbound selection of Secure Sockets Layer configurations.

Another way to accomplish this same result is to associate the SSL configuration with the ADMIN_SOAP endpoint for each individual process, deployment manager, Node1, Node2, Node1Server1, Node1Server2, Node2Server1, and Node2Server2. However, it is recommended that you use dynamic outbound selection because it is more efficient when defining a basic SSL configuration, its keystores, and its truststores at the cell scope. The example shows how to apply the node scope association, but the same principles apply for node groups, clusters, servers, and endpoints.

Note: If your topology includes clusters that span nodes or if your applications need to communicate between nodes, the configuration that is shown in the example does not work.

Certificate management using iKeyman prior to SSL

Starting in WebSphere Application Server Version 6.1, you can manage your certificates from the administrative console. When using versions of WebSphere Application Server prior to Version 6.1, use iKeyman for certificate management. iKeyman is a key management utility.

WebSphere Application Server certificate management requires that you define the keystores in your WebSphere Application Server configuration. With iKeyman, you need access to the keystore file only. You can read a keystore file with personal certificates and signers that is created in iKeyman. A keystore file can be read into the WebSphere Application Server configuration by using the **createKeyStore** command.

The majority of certificate management functions are the same between WebSphere Application Server and iKeyman, especially for personal certificates and signer certificates. However, certificate requests are special. The underlying behavior is different in the two certificate management schemes. Because of the different behavior, when a certificate request is generated from iKeyman, the process must be completed in iKeyman. For example, a certificate that is generated by a certificate request that originated in iKeyman must be received in iKeyman as well.

The same is true for WebSphere Application Server. For example, when a certificate is generated from a certificate request that originated in WebSphere Application Server, the certificate must be received in WebSphere Application Server.

You can perform the following certificate operations using iKeyman:

Table 88. Available certificate operations using iKeyman. This table describes the available certificate operations using iKeyman.

Types of certificates	Functions	Description
Personal certificates	Create a self-signed certificate	Creates a self-signed certificate and stores it in a keystore.
	List personal certificates	Lists all the personal certificates in a keystore.
	Get information about a personal certificate	Gets information about a personal certificate.
	Delete a personal certificate	Deletes a personal certificate from a keystore.
	Import a certificate	Imports a certificate from a keystore to a keystore.
	Export a certificate	Exports a certificate from a keystore to another keystore.

Table 88. Available certificate operations using iKeyman (continued). This table describes the available certificate operations using iKeyman.

Types of certificates	Functions	Description
	Extract a certificate	Extracts the signer part of a personal certificate to a file.
	Receive a certificate	Reads a certificate that comes from a certificate authority (CA) into a keystore.
Signer certificates	Add a signer certificate	Adds a signer certificate from a file to a keystore.
	List signer certificates	Lists all the signer certificates in a keystore.
	Get information about a signer certificate	Gets information about a signer certificate.
	Delete a signer certificate	Deletes a signer certificate from a keystore.
	Extract a signer certificate	Extracts a signer certificate from a keystore, and stores the certificate in a file.
Certificate requests	Create a certificate request	Creates a certificate request that can be sent to a CA.
	List certificate requests	Lists the certificate requests in a keystore.
	Get information about a certificate request	Gets information about a certificate request.
	Delete a certificate request	Deletes a certificate request from a keystore.
	Extract a certificate request	Extracts a certificate request to a file.

Certificate management in SSL

You can manage certificate operations that involve personal certificates, signer certificates, and personal certificate requests on the administrative console.

Types of certificates

WebSphere Application Server uses the certificates that reside in keystores to establish trust for a Secure Sockets Layer (SSL) connection. Click **Security > SSL certificate and key management > Manage endpoint security configurations > Inbound | Outbound > *SSL_configuration_name* > Key stores and certificates**, then select an existing or create a new keystore. After selecting a keystore, and depending on the type of certificate you need, choose one of the following types of certificates under Related Items:

- Personal certificate
- Signer certificate
- Certificate Authority (CA) certificates
- Personal certificate request

Table 89. Certificate operations. The following table describes the certificate operations that you can perform on the administrative console

Types of certificates	Functions	Description
Personal certificates	Create a self-signed certificate	Creates a self-signed certificate and stores it in a keystore.
	List personal certificates	Lists all the personal certificates in a keystore.
	Get information about a personal certificate	Gets information about a personal certificate.
	Delete a personal certificate	Deletes a personal certificate from a keystore.

Table 89. Certificate operations (continued). The following table describes the certificate operations that you can perform on the administrative console

Types of certificates	Functions	Description
	Import a certificate	Imports a certificate from a keystore to a keystore.
	Export a certificate	Exports a certificate from a keystore to another keystore.
	Extract a certificate	Extracts the signer part of a personal certificate to a file.
	Exchange signer certificates	Exchange signer part of a personal certificate between key store.
	Receive a certificate	Reads a certificate that comes from a certificate authority (CA) into a keystore.
	Replace a certificate	Replaces all occurrences of a personal certificate alias in the WebSphere Application Server configuration with another certificate. Also, replaces all occurrences of the personal certificates signer with the new personal certificate signer.
	Create a chained certificate	Creates a chained certificate and stores it in a keystore.
	Renew a certificate	Renews a certificate with a new public/private key pair and stores it in a keystore.
	Request a CA certificate	Makes a request to a CA using a CA client to obtain a CA certificate.
Certificate authority (CA) certificates	Create CA certificate	Sends a certificate request to an external certificate authority (CA).
	Revoke CA certificate	Sends a revocation request to an external certificate authority (CA).
Signer certificates	Add a signer certificate	Adds a signer certificate from a file to a keystore.
	List signer certificates	Lists all the signer certificates in a keystore.
	Get information about a signer certificate	Gets information about a signer certificate.
	Delete a signer certificate	Deletes a signer certificate from a keystore.
	Extract a signer certificate	Extracts a signer certificate from a keystore, and stores the certificate in a file.
	Retrieve a signer from a port	Retrieves a signer certificate from a port, and stores it in a key store.
Certificate requests	Create a certificate request	Creates a certificate request that can be sent to a CA.
	List certificate requests	Lists the certificate requests in a keystore.
	Get information about a certificate request	Gets information about a certificate request.
	Delete a certificate request	Deletes a certificate request from a keystore.
	Extract a certificate request	Extracts a certificate request to a file.

Personal certificates

Table 90. Personal certificate operations. The following table lists the operations that you can perform on personal certificates, the AdminTask object that you can use to perform that operation, and how to navigate to the certificate on the console:

Function	AdminTask object	Administrative console
Create a self-signed certificate	createSelfSignedCertificate	**Security > Secure Communications > Key store and certificates >** *key store* **> Create a Self-Signed Certificate**
List personal certificates	listPersonalCertificates	**Security > Secure Communications > Key store and certificates >** *key store* **> personal certificates**
Get information about a personal certificate	getPersonalCertificate	**Security > Secure Communications > Key store and certificates >** *key store* **> personal certificates >** *alias*
Delete a personal certificate	deletePersonalCertificate	**Security > Secure Communications > Key store and certificates >** *key store* **> personal certificates > delete**
Import a certificate	importCertificate	**Security > Secure Communications > Key store and certificates >** *key store* **> personal certificates > import**
Export a certificate	exportCertificate	**Security > Secure Communications > Key store and certificates >** *key store* **> personal certificates > export**
Extract a certificate	extractCertificate	**Security > Secure Communications > Key store and certificates >** *key store* **> personal certificates > extract**
Exchange signer certificates	exchangeSignerCertificates	**Security > Secure Communications > Key store and certificates > Exchange signers**
Create a chained certificate	createChainedCertificate	**Security > SSL certificate and key management > Key store and certificates >** ***keystore name*** **> Personal certificates**. Click **Create** button and select **Chained certificate**
Renew a certificate	renewChainedCertificate	**Security > SSL certificate and key management > Key store and certificates >** ***keystore name*** **> Personal certificates**. Select a certificate. Click **Renew** button.
Create a chained Certificate	createChainedCertificate	**Security > Secure communications > Key store and certificates > keystore > Create a chained certificate**.
Request a CA certificate	requestCACertificate	**Security > Secure communications > Key store and certificates > keystore > Request a CA certificate**.

Certificate authority (CA) certificates

Table 91. CA certificate operations. The following table lists the operations that you can perform on CA certificates, the AdminTask object that you can use to perform that operation, and how to navigate to the certificate on the console:

Function	AdminTask object	Administrative console
Create a CA certificate	createCACertificate	**Security > Secure Communications > Key store and certificates >** *key store* **> Personal certificates > Create > CA-signed certificate**
Revoke a CA certificate	revokeCACertificate	**Security > Secure Communications > Key store and certificates >** *key store* **> Personal certificates** *personal certificate* **> Revoke**

Signer certificates

Table 92. Signer certificate operations. The following table lists the operations that you can perform with signer certificates, the AdminTask object that you can use to perform the operation, and how to navigate to the certificate on the console:

Function	AdminTask object	Administrative console
Add a signer certificate	addSignerCertificate	**Security > Secure communications > Key store and certificates >** *key store* **> signer certificates > Add**
List signer certificates	listSignerCertificates	**Security > Secure communications > Key store and certificates >** *key store* **> signer certificates**
Get information about a signer certificate	getSignerCertificate	**Security > Secure communications > Key store and certificates >** *key store* **> signer certificates >** *alias*
Delete a signer certificate	deleteSignerCertificate	**Security > Secure communications > Key store and certificates >** *key store* **> signer certificate >delete**
Extract a signer certificate to a file	extractSignerCertificate	**Security > Secure communications > Key store and certificates >** *key store* **> signer certificates > extract**
Retrieve a signer certificate from a port	retrieveSignerFromPort	**Security > Secure communications > Key store and certificates >** *key store* **> signer certificates > retrieve from port**

Personal certificate requests

Table 93. Personal certificate request operations. The following table lists the operations that you can perform on personal certificate requests, the AdminTask object that you can use to perform that operation, and how to navigate to the certificate request on the console:

Function	AdminTask object	Administrative console
Create a personal certificate request	createCertificateRequest	**Security > Secure communications > Key store and certificates >** *key store* **> Personal certificate Requests > Add**
List personal certificate requests	listCertificateRequests	**Security > Secure communications > Key store and certificates >** *key store* **> Personal certificate requests**
Get information about a personal certificate request	getCertificateRequest	**Security > Secure communications > Key store and certificates >** *key store* **> Personal certificate requests >** *alias*
Delete a personal certificate request	deleteCertificateRequest	**Security > Secure communications > Key store and certificates >** *key store* **> Personal certificate requests > delete**
Extract a personal certificate request to a file	extractCertificateRequest	**Security > Secure communications > Key store and certificates >** *key store* **> Personal certificate requests > Extract**

Using the retrieveSigners command in SSL to enable server to server trust

You can add a signer certificate to a server's `trust.p12` file, allowing that server to securely communicate with another server. This can be done using the **retrieveSigners** command to add a signer to a server's `trust.p12` file after making changes to the `ssl.client.props` file.

Before you begin

The server that will be communicating as a client must be identified before the server to server trust can be established. You will make change to the ssl.client.props file and run the **retrieveSigners** command on the server communicating as a client. If both servers will be acting as a client , these steps will be required for both servers.

About this task

The ssl.client.props file is setup by default to configure Secure Socket Layer (SSL) communication for clients. This makes the default behavior of the **retrieveSigners** command work on the client's trust.p12 file and key.p12 file in the *profile_root*/etc directory. You can add a signer certificate to a server's trust.p12 file, allowing that server to act as a client communicating to another server. Using the **retrieveSigners** command to add a signer to a server's trust.p12 file requires some changes to the ssl.client.props file.

Procedure

1. Open the ssl.client.props file. The ssl.client.props file is located in *profile_root*/properties ditrectory.
2. Uncomment the section of ssl.client.props that starts with com.ibm.ssl.alias=AnotherSSLSettings property.
3. Uncomment the section of ssl.client.props that starts with com.ibm.ssl.trustStoreName=AnotherTrustStore property.
4. Enter the location of the trust store that the signer should be added. If you are using the server trust store for a deployment manager then it is located in *profile_root*/config/cells/*cell name*/trust.p12. If using the trust store for an application server, it is located in *profile_root*/config/cells/*cell name*/nodes/*node name*/trust.p12.
5. Update the remaining properties in this section with the values associated with the trust store being used. A description of the properties can be found in ssl.client.props client configuration file.
6. Optional: Uncomment and update section that starts with com.ibm.ssl.trustStoreName=AnotherKeyStore property. Most scenarios only require a signer to be added to the trust store. This example only adds a signer to the trust store, but you can also add a signer to the key store by updating the properties as you did for the trust store in steps 3 through 5.
7. Save the changes made to ssl.client.props.
8. Run the **retrieveSigners** command. For more information see the page about the retrieveSigners command.

```
retrieveSigners NodeDefaultTrustStore AnotherTrustStore -host ademyers.austin.ibm.com -port 8879
```

Example output:

```
CWPKI0308I: Adding signer alias "default_1" to local keystore
                    "AnotherTrustStore" with the following SHA digest:
                    F4:71:97:79:3E:C1:DC:E7:9F:8F:3D:F0:A0:15:1E:D1:44:73:2C:06
```

Results

After the steps have been successfully completed. the server acting as a client has the signing certificate of the other server. This allows that server to establish a SSL connection to the other server.

Example

The example shows the modified section of the ssl.client.props file assuming that the server's trust.p12 file is being used. Any trust store existing trust store can be used if the properties are provided for that trust store.

```
#-----------------------------------------------------------------------------
com.ibm.ssl.alias=AnotherSSLSettings
com.ibm.ssl.protocol=SSL_TLS
com.ibm.ssl.securityLevel=HIGH
```

```
com.ibm.ssl.trustManager=IbmX509
com.ibm.ssl.keyManager=IbmX509
com.ibm.ssl.contextProvider=IBMJSSE2
com.ibm.ssl.enableSignerExchangePrompt=true
#com.ibm.ssl.keyStoreClientAlias=default
#com.ibm.ssl.customTrustManagers=
#com.ibm.ssl.customKeyManager=
#com.ibm.ssl.dynamicSelectionInfo=
#com.ibm.ssl.enabledCipherSuites=

# KeyStore information
#com.ibm.ssl.keyStoreName=AnotherKeyStore
#com.ibm.ssl.keyStore=${user.root}/etc/key.p12
#com.ibm.ssl.keyStorePassword={xor}CDo9Hgw=
#com.ibm.ssl.keyStoreType=PKCS12
#com.ibm.ssl.keyStoreProvider=IBMJCE
#com.ibm.ssl.keyStoreFileBased=true

# TrustStore information
com.ibm.ssl.trustStoreName=AnotherTrustStore
com.ibm.ssl.trustStore=${user.root}/config/cells/localhostCell01/trust.p12
com.ibm.ssl.trustStorePassword={xor}CDo9Hgw=
com.ibm.ssl.trustStoreType=PKCS12
com.ibm.ssl.trustStoreProvider=IBMJCE
com.ibm.ssl.trustStoreFileBased=true
```

What to do next

After the signer has been added, edit the `ssl.client.props` file to comment out the sections that were to used to add the signer certificate.

Creating a Secure Sockets Layer configuration

Secure Sockets Layer (SSL) configurations contain the attributes that you need to control the behavior of client and server SSL endpoints. You create SSL configurations with unique names within specific management scopes on the inbound and outbound tree in the configuration topology. This task shows you how to define SSL configurations, including quality of protection and trust and key manager settings.

Before you begin

You must decide at which scope you need to define an SSL configuration, for instance, the cell, node group, node, server, cluster, or endpoint scope, from the least specific to the most specific scope. When you define an SSL configuration at the node scope, for example, only those processes within that node can load the SSL configuration; however, any processes at the endpoint in the cell can use an SSL configuration at the cell scope, which is higher in the topology.

You must also decide which scope to associate with the new SSL configuration, according to the processes that the configuration affects. For example, an SSL configuration for a hardware cryptographic device might require a keystore that is available only on a specific node, or you might need an SSL configuration for a connection to a particular SSL host and port. For more information, see “Dynamic outbound selection of Secure Sockets Layer configurations” on page 685.

gotcha: The security.xml file is restricted. Therefore, if you need to make changes to the security.xml file, verify that your user ID has administrator role authorization. If you are using a user ID with operator role authorization, you can perform a node synchronization, but any changes that you made to the security.xml file are not synchronized.

About this task

Complete the following steps in the administrative console:

Procedure

1. Click **Security > SSL certificate and key management > Manage endpoint security configurations**.

2. Select an SSL configuration link on either the Inbound or Outbound tree, depending on the process you are configuring.
   - If the scope is already associated with a configuration and alias, the SSL configuration alias and certificate alias are noted in parentheses.
   - If the parenthetical information is not included, then the scope is not associated. Instead, the scope inherits the configuration properties of the first scope above it that is associated with an SSL configuration and certificate alias.

   The cell scope must be associated with an SSL configuration because it is at the top of the topology and represents the default SSL configuration for the inbound or outbound connection.
3. Click **SSL configurations** under Related Items. You can view and select any of the SSL configurations that are configured at this scope. You can also view and select these configuration at every scope that is lower on the topology.
4. Click **New** to display the SSL configuration panel. You cannot select links under Additional Properties until you type a configuration name and click **Apply**.
5. Type an SSL configuration name. This field is required. The configuration name is the SSL configuration alias. Make the alias name unique within the list of SSL configuration aliases that are already created at the selected scope. The new SSL configuration uses this alias for other configuration tasks.
6. Select a truststore name from the drop-down list. A truststore name refers to a specific truststore that holds signer certificates that validate the trust of certificates sent by remote connections during an SSL handshake. If there is no truststore in the list, see “Creating a keystore configuration for a preexisting keystore file” on page 765 to create a new truststore, which is a keystore whose role is to establish trust during the connection.
7. Select a keystore name from the drop-down list. A keystore contains the personal certificates that represent a signer identity and the private key that WebSphere Application Server uses to encrypt and sign data.
   - If you change the keystore name, click **Get certificate aliases** to refresh the list of certificates from which you can choose a default alias. WebSphere Application Server uses a server alias for inbound connections and a client alias for outbound connections.
   - If there is no keystore in the list, see “Creating a keystore configuration for a preexisting keystore file” on page 765 to create a new keystore.
8. Choose a default server certificate alias for inbound connections. Select the default only when you have not specified an SSL configuration alias elsewhere and have not selected a certificate alias. A centrally managed SSL configuration tree can override the default alias. For more information, see “Central management of SSL configurations” on page 686.
9. Choose a default client certificate alias for outbound connections. Select the default only when the server SSL configuration specifies an SSL client authentication.
10. Review the identified management scope for the SSL configuration. Make the management scope in this field identical to the link you selected in Step 2. If you want to change the scope, you must click a different link in the topology tree and continue at Step 3.
11. Click **Apply** if you intend to configure Additional Properties. If not, go to Step 24.
12. Click **Quality of protection (QoP) settings** under Additional Properties. QoP settings define the strength of the SSL encryption, the integrity of the signer, and the authenticity of the certificate.
13. Select a client authentication setting to establish an SSL configuration for inbound connections and for clients to send their certificates, if appropriate.
    - If you select **None**, the server does not request that a client send a certificate during the handshake.
    - If you select **Supported**, the server requests that a client send a certificate. However, if the client does not have a certificate, the handshake might still succeed.
    - If you select **Required**, the server requests that a client send a certificate. However, if the client does not have a certificate, the handshake fails.

Important: The signer certificate that represents the client must be in the truststore that you select for the SSL configuration. By default, servers within the same cell trust each other because they use the common truststore, `trust.p12`, that is located in the cell directory of the configuration repository. However, if you use keystores and truststores that you create, perform a signer exchange before you select either **Supported** or **Required**.

14. Select a protocol for the SSL handshake.
    - The default protocol, SSL_TLS, supports client protocols TLSv1 and SSLv3.
    - The TLSv1 protocol supports TLS and TLSv1. The SSL server connection must support this protocol for the handshake to proceed.
    - SSLv2
    - SSLv3
    - The SSLv3 protocol supports SSL and SSLv3. The SSL server connection must support this protocol for the handshake to proceed.
    - TLS is TLSv1
    - TLSv1
    - SSL_TLSv2 is SSLv3 and TLSv1, TLSv1.1, TLSv1.2
    - TLSv1.1
    - TLSv1.2

    Important: Do not use the SSLv2 protocol for the SSL server connection. Use it only when necessary on the client side.
15. Select one of the following options:
    - A predefined Java Secure Socket Extension (JSSE) provider. The IBMJSSE2 provider is recommended for use on all platforms which support it. It is required for use by the channel framework SSL channel. When Federal Information Processing Standard (FIPS) is enabled, IBMJSSE2 is used in combination with the IBMJCEFIPS crypto provider.
    - A custom JSSE provider. Type a provider name in the **Custom provider** field.
16. Select from among the following cipher suite groups:
    - **Strong**: WebSphere Application Server can perform 128-bit confidentiality algorithms for encryption and support integrity signing algorithms. However, a strong cipher suite can affect the performance of the connection.
    - **Medium**: WebSphere Application Server can perform 40-bit encryption algorithms for encryption and support integrity signing algorithms.
    - **Weak**: WebSphere Application Server can support integrity signing algorithms but not to perform encryption. Select this option with care because passwords and other sensitive information that cross the network are visible to an Internet Protocol (IP) sniffer.
    - **Custom**: you can select specific ciphers. Any time you change the ciphers that are listed from a specific cipher suite group, the group name changes to Custom.
17. Click **Update selected ciphers** to view a list of the available ciphers for each cipher strength.
18. Click **OK** to return to the new SSL configuration panel.
19. Click **Trust and key managers** under Additional Properties.
20. Select a default trust manager for the primary SSL handshake trust decision.
    - Choose **IbmPKIX** when you require certificate revocation list (CRL) checking using CRL distribution points in the certificates or the online certificate status protocol (OCSP).
    - Choose **IbmX509** when you do not require CRL checking but do need increased performance. You can configure a custom trust manager to perform CRL checking, if necessary.
21. Define a custom trust manager, if appropriate. You can define a custom trust manager that runs with the default trust manager you select. The custom trust manager must implement the JSSE

javax.net.ssl.X509TrustManager interface and, optionally, the com.ibm.wsspi.ssl.TrustManagerExtendedInfo interface to obtain product-specific information.

a. Click **Security > SSL certificate and key management > Manage endpoint security configurations > *SSL_configuration* > Trust and key managers > Trust managers > New**.
b. Type a unique trust manager name.
c. Select the **Custom** option.
d. Type a class name.
e. Click **OK**. When you return to the Trust and key managers panel, the new custom trust manager displays in the **Additional ordered trust managers** field. Use the left and right list boxes to add and remove custom trust managers.

22. Select a key manager for the SSL configuration. By default, IbmX509 is the only key manager unless you create a custom key manager.

    Important: If you choose to implement your own key manager, you can affect the alias selection behavior because the key manager is responsible for selecting the certificate alias from the keystore. The custom key manager might not interpret the SSL configuration as the WebSphere Application Server key manager IbmX509 does. To define a custom key manager, click **Security > Secure communications > SSL configurations > *SSL_configuration* > Trust and key managers > Key managers > New**.

23. Click **OK** to save the trust and key manager settings and return to the new SSL configuration panel.
24. Click **Save** to save the new SSL configuration.

Results

Important: You can override the default trust manager when you configure at least one custom trust manager and set the com.ibm.ssl.skipDefaultTrustManagerWhenCustomDefined property to `true`. Click **Custom Property** on the SSL configuration panel. However, if you change the default, you leave all the trust decisions to the custom trust manager, which is not recommended for production environments. In test environments, use a dummy trust manager to avoid certificate validation. Remember that these environment are not secure.

What to do next

In this release of WebSphere Application Server, you can associate SSL configurations with protocols using one of the following methods:

- Set the SSL configuration on the thread programmatically
- Associate the SSL configuration with an outbound protocol, and target host and port. For more information, see “Associating a Secure Sockets Layer configuration dynamically with an outbound protocol and remote secure endpoint” on page 743
- Associate the SSL configuration directly using the alias. For more information, see “Selecting an SSL configuration alias directly from an endpoint configuration” on page 747
- Manage the SSL configurations centrally by associating them with SSL configuration groups or zones that are scoped for endpoints. For more information, see “Associating Secure Sockets Layer configurations centrally with inbound and outbound scopes” on page 746.

SSL certificate and key management

Use this page to configure security for Secure Socket Layer (SSL) and key management, certificates, and notifications. The SSL protocol provides secure communications between remote server processes or endpoints. SSL security can be used for establishing communications inbound to and outbound from an endpoint. To establish secure communications, a certificate and an SSL configuration must be specified for the endpoint.

To view this administrative console page, click **Security > SSL certificate and key management**.

Configuration settings
Specifies the following administrative console tasks:

- Manage endpoint security configurations
- Manage certificate expiration

Use Federal Information Processing Standard (FIPS) algorithms
Specifies the Federal Information Processing Standard (FIPS)-compliant Java cryptography engine is enabled.

- Does not affect the SSL cryptography that is performed by the application server for z/OS System Secure Sockets Layer (SSSL).
- Does not change the JSSE provider if this cell includes any Application Server versions before the application server for z/OS Version 6.0.x.

When you select the **Use the Federal Information Processing Standard (FIPS)** option, the Lightweight Third Party Authentication (LTPA) implementation uses IBMJCEFIPS. IBMJCEFIPS supports the Federal Information Processing Standard (FIPS)-approved cryptographic algorithms for Data Encryption Standard (DES), Triple DES, and Advanced Encryption Standard (AES). Although the LTPA keys are backwards compatible with prior releases of the application server, the LTPA token is not compatible with prior releases. In prior releases, the application server did not generate the LTPA token using a FIPS-approved algorithm.

The IBMJSSE2 JSSE provider does not perform cryptographic functions directly, and therefore does not need to be FIPS-approved. Instead, the IBMJSSE2 JSSE provider uses the JCE framework for cryptographic functions and uses IBMJCEFIPS when FIPS mode is enabled.

Important: HP-UX The IBMJSSEFIPS provider is not supported on the HP-UX platform. However, the IBMJSSE2 provider, which uses IBMJCEFIPS, is supported on the HP-UX platform.

Information	Value
Default:	Disabled

Dynamically update the runtime when SSL configuration changes occur
Specifies that all of the SSL-related attributes and LTPA keys that change must be read from the configuration dynamically after they have been saved, then reused for new connections. To avoid customer impact, it is recommended that changes to production servers be made during off-peak periods.

Information	Value
Default:	Enabled

When this option is selected, the configuration is updated each time you configure an SSL communication.

SSL configurations for selected scopes
Use this page to display Secure Socket Layer (SSL) configurations for selected scopes, such as a cell, node, server, or cluster. From this page you can navigate to configuration panels for the following: SSL configurations, dynamic inbound and outbound endpoint SSL configurations, key stores, key sets, key set groups, key managers, and trust managers.

To view this administrative console page, click **Security > SSL certificate and key management**. Under Configuration settings, click **Manage endpoint security configurations > {Inbound | Outbound} > *ssl_configuration***.

Name
Specifies the SSL configuration scope, which is derived from the selected object in the hierarchy.

Information	Value
Data type:	Text

Direction
Specifies the direction for which the SSLConfig applies. Inbound refers to any listener port. Outbound refers to outbound end point connections.

Information	Value
Data type:	Text

SSL configuration
Specifies the SSL configuration that is used by requests at this scope.

Information	Value
Data type:	Text

Update certificate alias list
Specifies the certificate aliases contained in the key store for this SSL configuration can be selected from the **Certificate alias in key store** list. You must update the certificate list after choosing a different SSL configuration alias. If you do not update the list, you will save a certificate alias that is not contained in the SSL configuration.

Manage certificates
Specifies to open the keystore panel for the key store in this SSL configuration, which enables you to manage personal certificates, signers, and certificate requests.

Certificate alias in key store
Specifies the certificate to use in the key store.

If you select **None**, the Java Secure Sockets Extension (JSSE) key manager determines which certificate is used. If multiple certificates exist in the key store, the key manager might not consistently select the same certificate.

Information	Value
Data type:	Text

SSL configurations collection
Use this page to define a list of Secure Sockets Layer (SSL) configurations.

To view this administrative console page, click **Security > SSL certificate and key management**. Under Configuration settings, click **Manage endpoint security configurations > {Inbound | Outbound} > *ssl_configuration***. Under Related items, click **SSL configurations**.

Table 94. SSL configurations buttons. This table lists the SSL configurations buttons.

Button	Resulting action
New	The Java Secure Socket Extension (JSSE) repertoire is for Java-based SSL communications. You can define a new JSSE configuration that can be used to create an SSLContext, URLStreamHandler, SSLSocketFactory, SSLServerSocketFactory, and so on, using the com.ibm.websphere.ssl.JSSEHelper API.
Delete	Deletes an existing JSSE configuration (administrator only). Be careful that any references to the SSL configuration have been removed prior to deleting this SSL configuration.

Name
Specifies the unique name of the SSL configuration in the management scope.

SSL configuration settings
Use this page to define Secure Sockets Layer (SSL) configuration properties.

To view this administrative console page, click **Security > SSL certificate and key management**. Under Configuration settings, click **Manage endpoint security configurations > {Inbound | Outbound} > *nodes name***. Under Related items, click **SSL configurations > New**.

Name
Specifies the unique name of the SSL configuration within the management scope in which it resides. For ways to programmatically access the properties that are configured for this SSL configuration, see the com.ibm.websphere.ssl.JSSEHelper application programming interface (API).

Information	Value
Data type:	Text

Trust store name
Specifies a reference to a specific trust store used by Java Secure Sockets Extension (JSSE). The trust store holds signer certificates that validate the trust of certificates sent by remote connections during an SSL handshake.

Information	Value
Data type:	Text
Default:	*selected trust store*

Key store name
Specifies a reference to a specific key store. The key store holds personal certificates that represent the identity of one side of a connection. The public key of this personal certificate is sent to the other side of the connection to establish trust during the handshake. The remote side of the connection needs the root certificate authority (CA) certificate or self-signed public key (signer) to be in the trust store to validate this personal certificate.

Information	Value
Data type:	Text
Default:	*selected key store*

Get certificate aliases
Queries the keystore for the aliases of all the personal certificates in the keystore from which to choose.

Default server certificate alias
Specifies the certificate alias used as the identity for this SSL configuration if one has not been specified elsewhere.

If you select **None**, the Java Secure Sockets Extension (JSSE) key manager determines which certificate is used. If multiple certificates exist in the key store, the key manager might not consistently select the same certificate.

Information	Value
Data type:	Text

Default client certificate alias
Specifies the certificate alias to be used if this configuration is to be used as a client.

If you select **None**, the Java Secure Sockets Extension (JSSE) key manager determines which certificate is used. If multiple certificates exist in the key store, the key manager might not consistently select the same certificate.

Information	Value
Data type:	Text

Management scope
Specifies the scope where this SSL configuration is visible. For example, if you choose a specific node, then the configuration is visible only on that node and on any servers that are part of that node.

Information	Value
Data type:	Text

Secure Sockets Layer client certificate authentication
Client software that wants to establish a secure connect to a server by using Secure Socket Layer (SSL) protocol initiates by leveraging SSL protocol or the enhanced protocol called Transport Layer Security (TLS) to perform a SSL handshake with SSL certificates. A personal certificate can represent the server or it can represent a particular client, and is signed by a Certificate Authority (CA) to ensure that the personal certificate is correctly identified.

SSL ensures that the administrator has the CA signer certificate available that is used to sign the personal certificate, and that it is stored in both the client and or the server trusted store. SSL client certificate authentication takes place during the connection handshake by using SSL certificates.

The following events must occur during this process:
- The server side must determine if client authentication is going to take place. The client authentication must be enabled in the SSL configuration of the server and the Common Secure Interoperability version 2 (CSIv2) configuration if Inter-ORB Protocol (IIOP) is used.
- The CSIv2 configuration must take place in global security, not in a security domain.
- The signer certificate of the client must be extracted from the key store of the client and added to the trust store of the server.
- The signer certificate of the server must to be extracted from the key store of the server and added to the trust store of the client.

Configuring a WebSphere server for client authentication

Client certificate authentication occurs if the server side requests that the client side send a certificate. A Websphere server can be configured for client certificate authentication on the SSL configuration. However, if client authentication is needed for IIOP then it must be configured on the CSIv2 configuration.

To configure client certificate authentication on the SSL configuration using the administrative console:
1. Click **Security** > **SSL certificate and key management** > **SSL configurations**.
2. Select a SSL configuration.
3. Under Additional Properties, select Quality of protection (QoP) settings.
4. Under Client authentication, select **Required**.
5. Click **OK** to save the changes.

Note: You can also use the modifySSLConfig command with the -clientAuthentication flag set to `true` to enable client authentication. See SSLConfigCommands command group for the AdminTask object for more information about this command.

.

To configure client certificate authentication on a CSIv2 inbound connection using the administrative console:

1. Click**Security** > **Global Security**.
2. Under RMI/IIOP security, select **CSIv2 inbound communications**.
3. In the CSIv2 Transport Layer section, and under Client certificate authentication, select **Required**.
4. Click **OK** to save the changes

Note: You can also use the configureCSIInbound command with the -clientCertAuth flag set to `Required` to enable client authentication on CSIv2. Read SecurityConfigurationCommands command group for the AdminTask object for more information about this command.

If the client side is set up for client authentication, the signer certificate of the client must be added to the trust store of the server. When you have a certificate from the client in a certificate file it can be added to the trust store of the server.

To add a signer to the trust store of the server using the administrative console:

1. Click **Security** > **SSL certificate and key management** > **Key stores and certificates**.
2. Select the trust store that is configured for client authentication.
3. Under Additional Properties, select Signer Certificates.
4. Click **Add**.
5. In the Alias field, type an alias name under which to store the certificate.
6. In the File name box, type the full path to the certificate file.
7. Click **OK** to save the changes

Note: You can also use the addSignerCertificate command to add a signer to the trust store of the server. Read SignerCertificateCommands command group for the AdminTask object for more information about this command.

Note: If you are using client authentication in a cluster environment, client authentication must be configured for each node that the servers in the cluster are located in.

Setting up the client side for client authentication

Clients:

Administrative clients, thin clients or pure clients must have a personal certificate in their key stores. The WebSphere client default key store that is created when WebSphere Application Server is installed already has a personal certificate in it. This key store can be found in the `ssl.client.props` file in the com.ibm.ssl.keyStore property. The client key stores are not managed by WebSphere Application Server, so the Key Management utility (iKeyman) or Java keytool utility can be used to extract the certificate to a certificate file.

To extract a certificate using iKeyman:

1. Start iKeyman.
2. Select **Key Database File** > **open**.
3. Enter the path to the keystore file. You can obtain this from the `ssl.client.props` file.
4. Click **OK**.

5. Enter the password to the key store and click **OK**
6. Under Personal Certificates, select the client default certificate.
7. Enter a path and file name for the certificate file and click **OK**.

The file that contains the extracted certificate can be used to add the signer to the trust store of the server. Follow the steps in the “Configuring a WebSphere server for client authentication” section to add that signer to the server trust store.

If the communication is over IIOP, the following properties must be set in the `sas.client.props` file.

- Enable SSL:

```
com.ibm.CSI.performTransportAssocSSLTLSSupported=true
com.ibm.CSI.performTransportAssocSSLTLSRequired=false
```

- Disable client authentication at the message layer:

```
com.ibm.CSI.performClientAuthenticationRequired=false
com.ibm.CSI.performClientAuthenticationSupported=false
```

- Enable client authentication at the transport layer (this is supported, but not required):

```
com.ibm.CSI.performTLClientAuthenticationRequired=false
com.ibm.CSI.performTLClientAuthenticationSupported=true
```

Thin clients and pure clients might not use the WebSphere Application Server SSL properties file, `ssl.client.props`. They most likely use the Java system properties to set the client key store and trust store. The signer certificate of the server must be added to the trust store that is specified with the java.net.ssl.trustStore system property. Keytool or iKeyman can be used to add the signer certificate. The signer must be extracted from the personal certificate in the key store specified by the javax.net.ssl.keyStore system property, and added to the trust store of the server.

For example:

```
javax.net.ssl.keyStore
javax.net.ssl.keyStorePassword
javax.net.ssl.keyStoreType
javax.net.ssl.trustStore
javax.net.ssl.trustStorePassword
javax.net.ssl.trustStoreType
```

Server acting as a client:

The client can be a WebSphere server acting as a client. If so, determine which SSL configuration is being used as the client side of the communication, extract it's certificate's signer and add it to the server side trust store. It is recommended that the root certificate signer be used.

To extract the root certificate using the administrative console:

1. Click Security > SSL certificate and key management > Key stores and certificates.
2. Under the Keystore usages pull-down, select **Root certificate keystore**.
3. Select either **DmgrDefaultRootStore** (for a network deployment server) or **NodeDefaultRootStore** (for an application server).
4. Under Additional Properties, select **Personal certificates**.
5. Select the default root certificate (usually called **root**), and then click **Extract**.
6. In the Certificate file name box, type a full path to the file in which to hold the certificate.
7. Click **OK** to save.

Note: You can also use the extractCertificate command to extract the root certificate. Read PersonalCertificateCommands command group for the AdminTask object for more information about this command.

The certificate file that is created can be carried to the server side and added to the trust store of the server.

When a server acts as a client, the client side server requires the signer from the destination server. The signer can be retrieved using the signer certificate `Retrieve from port` option.

To retrieve the signer from the port using the administrative console:
1. Click **Security** > **SSL certificate and key management** > **Key stores and certificates**.
2. Select the trust store of the server from the collection.
3. Under Additional Properties, select **Signer certificates**.
4. Click **Retrieve from port**.
5. Enter a destination host name and a destination port name.
6. Enter an alias name for the certificate.
7. Click **Retrieve signer information**.
8. Click **OK** to save.

You can also use the retrieveSignerFromPort command to retrieve the signer from the port. Read SignerCertificateCommands command group for the AdminTask object for more information about this command.

Setting up a browser for client authentication:

When WebSphere Application Server is configured for client certificate authentication, and an attempt is made to access the server from a browser, the browser must have a certificate for the client certificate authentication. If the default SSL configuration of the server was modified to enable client certificate authentication you are unable to login to the administrative console.

You can create a certificate for the browser by using the administrative console. You must first create a key store and then create a chained certificate. After the certificate is created, use the instructions for your browser to import a certificate. Browsers require that each part of the chain be added to verify the certificate, so the root certificate must be extracted and added to the browser. Follow the instructions in the "Setting up the client side for client authentication" section for information about extracting the root certificate.

To create a key store using the administrative console:
1. Click **Security** > **SSL certificate and key management** > **Key stores and certificates**.
2. Click **New**.
3. Enter a name for the key store.
4. Enter the full path to the key store file.
5. Enter a password for the key store and then confirm.
6. Click **OK** to save.

To create a chained certificate using the administrative console:
1. Click **Security** > **SSL certificate and key management** > **Key stores and certificates**.
2. Select the key store you created previously.
3. Under Additional Properties, click **Personal certificates**.
4. In the pull-down list under the Create button, select **Chained Certificate**.
5. Enter an alias name for the certificate.
6. Provide a common name for the certificate. The name is the "CN=" part of the subject DN.
7. You can enter information in any of the remaining fields to build the subject DN of the chained certificate.

8. Click **OK** to save.

You can also use the createKeyStore command to create a key store. Read KeyStoreCommands command group for the AdminTask object for more information about this command.

You can also use the createChainedCertificate command to create a chained certificate. Read PersonalCertificateCommands command group for the AdminTask object for more information about this command.

Note: When client certificate authentication is enabled, web certificate authentication can then be performed as discussed in the next section.

Web certificate authentication

Certificate base authentication can be performed on Java 2 Platform, Enterprise Edition (J2EE) web modules when the module is configured for client certificate authentication. This enables a user to login to a web module using a certificate to authenticate, and to map that certificate to a user from the registry.

Enabling web certificate authentication requires that the SSL configuration of the server be configured for client certificate authentication on the server where the module is installed.

The server side determines that client authentication is to take place. See the “Configuring a WebSphere server for client authentication” section for information about how to configure client authentication. The client side must have the signer from the server to add to the client truststore. See the “Setting up the client side for client authentication” section for more information.

The `web.xml` file of the web module must have the authentication method set to CLIENT-AUTH in the login-config section of the `web.xml` file:

```
<login-config>
<auth-method>CLIENT-CERT</auth-method>
</login-config>
```

The certificate must map to a user in the registry or you are unable to login to that web module.

For localOS user registries, the CN value of the certificate subject DN must map to a user in the local OS user registry. For example, if the certificate subject DN is *CN=tester,o=ibm,c=us*, then `tester` is the user searched for in the local user registry. If that user does not exist in the local registry then the authentication fails.

The Lightweight Directory Access Protocol (LDAP) user registry provides more options for mapping a certificate to a user identity. The default certificate mapping mode in LDAP is used for an exact DN match between the entry in the LDAP registry and the subject DN in the certificate. For example, if the certificate DN is *CN=user1,o=ibm,c=us*, then there must be an entry in the LDAP registry with that exact value. The LDAP user registry also has a certificate filter option that can provide a match to a particular part of the certificate subject DN against entries in the LDAP repository. For more detail on LDAP certificate mapping, read “Lightweight Directory Access Protocol repository configuration settings”.

In a federated repositories configuration, by default, client certificate login is not supported for the file-based repository. To enable support for certificate mapping in the file-based repository follow the procedure in the topic, Enabling client certificate login support in a federated repositories file-based repository.

The federated repository LDAP registry supports certificate mapping. It uses the same mapping rules and properties that the LDAP user registry uses.

Custom user registry can map certificates to a user if the custom registry implemented the mapCertificate() method.

Certificate authority (CA) client configuration

Use this page to create, modify, and configure a certificate authority (CA) client.

To view this administrative console page, click **Security** > **SSL Configurations and key management** . Under Related Items, click **Certificate Authority (CA) client configurations**. Then click either the **New** button or select an existing CA client by clicking on its *<client_name>*.

Name

Specifies the unique name of the CA client configuration. This is the name to identify the CA client object. This name needs to be unique to the scope.

Information	Value
Data type:	String

Implementation class

Specifies the name of the module that implements the com.ibm.wsspi.ssl.WSKPIClient interface that is used to act as a client to a CA. This implementation class connects to the CA server and performs a certificate create, revoke, or replace.

Information	Value
Default:	String

CA server host name

Specifies the host name of the CA server, if the implementation requires a host name.

Information	Value
Data type:	String

Port

Specifies the port where the CA server will communicate, if the implementation requires a port.

Information	Value
Data type:	String

User name

Specifies the user Id used to connect to the CA server, if the implementation requires a user to login to the CA.

Information	Value
Data type:	String

Password

Specifies the password for the connection to the CA server.

Information	Value
Data type:	String

Confirm password
Confirms the password that is provided in the password field.

Information	Value
Data type:	String

Number of times to poll
Specifies the number of times to check the CA server to see if the certificate is complete. This poll number applies to the CA that does not return certificates right away.

Information	Value
Default:	5

Polling interval when requesting certificates
Specifies the amount of time, in minutes, between checks to the CA server to see if the certificate is complete.

Information	Value
Default:	10

Custom properties
Specifies arbitrary name and value pairs of data. The name is a property key, and the value is a string value that can be used to set internal system configuration properties.

Information	Value
Data type:	string

Certificate authority (CA) client configuration collections
Use this page to define and manage certificate authority (CA) clients or view and modify existing CA clients.

This panel allows you to create a certificate authority (CA) client object in the configuration. You can also view and modify existing CA clients. The information in the CA client object can then be used by the runtime to connect to a CA server to request, revoke, or query a certificate.

To view this administrative console page, click **Security** > **SSL Configurations and key management**. Under Related Items, click **Certificate Authority (CA) client configurations**.

Table 95. CA client configuration buttons.

This table describes the CA client configuration buttons.

Button	Resulting action
New	Adds a new CA client object that can be referenced by Secure Sockets Layer (SSL) configurations.
Delete	Deletes an existing CA client object.

Name
Identifies the unique name of the CA client configuration.

Implementation class
Identifies the name of the module that implements the **com.ibm.wsspi.ssl.WSKPIClient** interface that is used to act as a client to a CA.

Management Scope
Identifies the scope where this secure sockets layer (SSL) configuration is visible.

Creating a chained personal certificate in SSL
A chained personal certificate is a personal certificate that is created by using another personal certificate to sign it. This chaining allows a certificate to be signed with a certificate (a root certificate) that has a long life span. Root certificates are stored in the **DmgrDefaultRootStore** or **NodeDefaultRootStore**. The server's default personal certificate is a chained certificate created when the profile is created. Chained certificates can also be created after profile creation

Before you begin

You use the administrative console to create a chained personal certificate.

Procedure
1. Click **Security > SSL certificate and key management**.
2. Under Related Items, click **Key stores and certificates**.
3. Click a **<keystore name>** to which you want to add the chained personal certificate.
4. Under Additional Properties, click **Personal certificates** .
5. Click the **Create** button and select **Chained Certificate** The **listCertificates** AdminTask can be used to generate the list of root certificates available to sign the certificate.
6. Fill in the following information to the General Properties section as follows:
   - Supply an alias name.
   - Select Root certificate from the pull down list.
   - Key size
   - Common name
   - Validity period
   - Organization
   - Organization Unit
   - Locality
   - State/Province
   - Zip code
   - Country or region
   -
7. Click **Apply** then **OK**.

Results

The certificate is created, signed by the root certificate specified, and stored in the keystore. Once a chained personal certificate is created, the certificate can be used by the runtime for SSL communication.

Recovering deleted certificates in SSL
The SSL configuration contains a keystore created to hold personal certificates that were deleted from other keystores in the configuration. Perform this task to recover deleted certificates.

Before you begin

The SSL configuration contains a keystore created to hold personal certificates that were deleted from other keystores in the configuration. On a stand alone application server the keystore is called `NodeDefaultDeletedStore` and on a deployment manager the keystore is called `DmgrDefaultDeletedStore`.

When a personal certificate is deleted from a keystore using the administrative console or in a script using the deleteCertificate AdminTask, a copy of the certificate is stored in the `DmgrDeletedKeyStore` or `NodeDeletedKeyStore`. The personal certificate takes the alias of <keystore>_<alias> > in the deleted keystore. If the alias name is already used in that deleted keystore a <unique number> is appended to the alias.

A personal certificate can be recovered from the deleted keystore by importing or exporting the personal certificate to a keystore in the configuration. To recover a personal certificate using the administrative console perform the following steps:

Procedure

1. Click **Security > SSL certificate and key management**.
2. Under Related Items, click **Key stores and certificates**.
3. From the Keystore usages drop-down list, select **Deleted certificates keystore**.
4. Click **DmgrDefaultDeletedStore** or **NodeDefaultDeletedStore**.
5. Under Additional Properties, click **Personal certificates**.
6. Select a certificate.
7. Select **Export**
8. Click **OK**.
9. Perform the following:
   - • Enter the keystore password of the deleted keystore.
   - • Enter The alias to be assigned to the certificate (in the key store that will receive the certificate).
   - • Select the ‘Managed key store’ radio button.
   - • Select the key store from the drop down list that will receive the certificate.
   - Click **Apply** then **OK**.

Results

Note: To recover a personal certificate you can also use the exportCertToManagedKS AdminTask command.

Renewing a certificate in SSL

If a personal certificate has been compromised or is about to expire, then it should be renewed. Renewing a certificate recreates the certificate with all the information from the original certificate, but with a new expiration period and public/private key pair. Only self-signed certificates and chained certificates created by WebSphere can be renewed. If the certificate used to sign the chained certificate is not in the root keystore then the default root certificate is used to renew the certificate.

Before you begin

You use the administrative console to renew the certificate.

Procedure

1. Click **Security > SSL certificate and key management**.
2. Under Related Items, click **Key stores and certificates**.
3. Click the appropriate **<keystore name>** to which you want to add the new certificate.

   Note: Only self-signed certificates and chained certificates signed with root certificates from the root keystore can be renewed.
4. Under Additional Properties, click **Personal certificates** to list the personal certificates.
5. Select a personal certificate from the list.

6. Click the **Renew** button.
7. Click **Apply** then **OK**.

Results

The certificate is renewed in the key store selected in the path to this panel . If the certificate is not a self-signed certificate or a chained certificate signed with a root certificate from the default root store, an error is returned.

Note: If this command is used with a CA certificate, an error occurs.

Revoking a CA certificate in SSL

If a certificate authority (CA) certificate is compromised and the servers cannot trust it anymore that CA certificate can be revoked. To revoke a CA certificate, you perform the following task.

Before you begin

You use the administrative console to replace or revoke a CA certificate.

Procedure

1. Click **Security > SSL certificate and key management**.
2. Under Related Items, click **Key stores and certificates**.
3. Click a **<keystore name>** to which you want to add the new CA certificate.
4. Under Additional Properties, click **Personal certificates** to list the personal certificates.
5. Select a certificate to revoke (a CA certificate)
6. Click the **Revoke** button.
7. Fill in the following information to the CA certificate section.
   - Revocation password
   - Revocation reason
8. Click **Apply** then **OK**.

Results

The certificate is revoked in the key store selected in the path. If the certificate selected was not a CA certificate, then an error is returned.

What to do next

Using a CA client to create a personal certificate to be used as the default personal certificate

An external certificate authority (CA) certificate can be used as the server default personal certificate. The CA certificate can be created using a CA client.

Before you begin

What you need to have before you perform this task is as follows:

- A certificate authority (CA) to make the certificate request to.
- A module that implements the **com.ibm.wsspi.ssl.WSPKIClient** interface. This module is needed to connect to the CA server and request a certificate.

You use the administrative console to view or modify a CA client.

Procedure

1. Click **Security > SSL certificate and key management**.
2. Under Related Items, click **Certificate Authority (CA) client configurations**. A panel displaying the existing CA clients appears.
3. Click the **New** button.
4. Enter the CA client information as required.
   - Name of the CA client.
   - The management scope (selected from the drop-down list.
   - Implementation class.
   - CA server host name.
   - User name.
   - Password.
   - Confirm of password.
   - Number of times to poll.
   - Polling interval (in minutes) when requesting certificates.
   - Custom properties.
5. Click **Apply** then **Save**.
6. Navigate to the Server default key store personal certificate. **Security > SSL configuration and certificate management > Key stores and certificates > <server_default_keystore>** . Under Additional properties, click **Personal certificates**
7. Click the **Create** button and select **CA-signed certificate**
8. Fill in the following information to the CA certificate section.
   - Revocation password
   - Confirm password.
   - Select the CA client that applies to this CA certificate.

     Note: You can create a new CA client to apply to this CA authority by clicking the **New** button.
   - Fill in the following information to the Request Specification section:
     – Select the radio button for Predefined request alias if you have a predefined alias.
     – If you do not have a predefined alias, fill in the following fields:
       - Type an alias name in the Alias field. The alias identifies the certificate request in the keystore.
       - Type a common name (CN) value. This value is the CN value in the certificate distinguished name (DN).
       - **Optional:** Type an organization value. This value is the O value in the certificate DN.
       - **Optional:** Select a key size value. The valid key size values are 512, 1024, 2048, 4096, and 8192. The default key size value is 2048 bits.
       - Locality
       - **Optional:** Type the State or Province value. This value is the ST value in the certificate DN.
       - **Optional:** Type a zip code value. The zip code value is the POSTALCODE value in the certificate DN.
       - **Optional:** Type a country or region value from the list. This country value is the C= value in the certificate request DN.
       - Validity period
9. Click **Apply** then **Save**.

10. Navigate to the Server Default Key store's personal certificates **Security > SSL configuration and certificate management > Key stores and certificates > <server_default_keystore>** . Under Additional properties, click **Personal certificates**
11. Select the server default personal certificate and click the **Replace** button.
12. Select the CA certificate alias from the list of aliases.
13. Click **Apply** then **Save**.

Results

The CA certificate alias replaces the alias of the default certificate in places where it is referenced in the configuration. All signer certificates from the default certificate are replaced with the signer certificate from the CA certificate.

Creating a CA certificate in SSL

Certificates can be created by a certificate authority (CA) when a CAClient object is configured to connect to the CA to create the certificate. Certificates created by a certificate authority (CA) with a CA client are tracked in the security configuration in an object called `CACertificate`. The certificate is stored in a keystore and a `CACertificate` object is added to the configuration to reference the certificate. CA certificates are personal certificates.

Before you begin

Before you begin, a CA client must be created to connect to the CA server. You then use the administrative console to create a CA certificate.

Note: In this release of WebSphere Application Server, the valid key size values are 512, 1024, 2048, 4096, and 8192. The default key size value is 2048 bits.

Procedure

1. Click **Security > SSL certificate and key management**.
2. Under Related Items, click **Key stores and certificates**.
3. Click a **<keystore name>** to which you want to add the new CA certificate.
4. Under Additional Properties, click **Personal certificates** to create a new CA certificate in the configuration.

   Note: You can also create a CA certificate by using the **requestCACertificate** AdminTask .
5. Click the **Create** button and select **CA-signed Certificate**
6. Fill in the following information to the CA certificate section.
   - Revocation password
   - Confirm password.
   - Select the CA client from the pull down list.

     Note: You can create a new CA client to apply to this CA authority by clicking the **New** button.
   - Fill in the following information to the Request Specification section:
     – Select the radio button for a predefined request alias if a certificate request is already created.
     – If you do not have a predefined certificate request alias, fill in the following fields:
       a. Type an alias name in the Alias field. The alias identifies the certificate request in the keystore.
       b. Type a common name (CN) value. This value is the CN value in the certificate distinguished name (DN).
       c. **Optional:** Type an organization value. This value is the O value in the certificate DN.

d. **Optional:** Select a key size value. The valid key size values are 512, 1024, 2048, 4096, and 8192. The default key size value is 2048 bits.

e. Locality

f. **Optional:** Type the State or Province value. This value is the ST value in the certificate DN.

g. **Optional:** Type a zip code value. The zip code value is the POSTALCODE value in the certificate DN.

h. **Optional:** Type a country or region value from the list. This country value is the C= value in the certificate request DN.

7. Click **Apply** then **OK**.

Results

The certificate is stored in the keystore selected in the path to this panel and a CACertificate configuration object is created. Once a CA certificate is created the certificate can be used by the runtime for SSL communication.

An existing certificate request can be used to create the CA certificate or a new certificate request can be created. This panel uses the requestCAClient AdminTask to create the CA certificate.

Developing the WSPKIClient interface for communicating with a certificate authority

Implementing the WSPKIClient interface enables WebSphere Application Server security to communicate with a remote certificate authority (CA).

Procedure

1. Initialize the WSPKIClient method, with init(java.util.HashMap).

   ```
   public void init(java.util.HashMap initAttrs) throws WSPKIException;
   ```

   This method is called by WebSphere Application Server runtime to set up connection information to a CA.

2. • Request a certificate with requestCertificate(byte[], X500Principal, byte[], java.util.HashMap).

   ```
   public X509Certificate[] requestCertificate(byte[] certReq,
   X500Principal SubjectDN, byte[] revocationPassword,
   java.util.HashMap customAttrs) throws WSPKIException;
   ```

   This method is called by WebSphere Application Server runtime to connect to a CA and requests a certificate signed by the authority. A X509Certificate[] is returned if the requested certificate is created. If a null is returned then queryCertificate() is called to check if the certificate is ready. This method is used when the CA requires manual intervention to process a certificate request.

   You can invoke this operation from the administrative console using the “Creating a CA certificate in SSL” on page 731 task and from a client using the requestCertificate script.

3. • Revoke a certificate with revokeCertificate(X509Certiifcate[], byte[], String, java.util.HashMap).

   ```
   public void revokeCertificate(X509Certificate[] cert, byte[] revocationPassword,
   String revocationReason, java.util.HashMap customAttrs) throws WSPKIException;
   ```

   This method called by WebSphere Application Server runtime to submit a request to a CA to revoke a certificate.

   You can invoke this operation from the administrative console using the revoke CA certificate task, “Revoking a CA certificate in SSL” on page 729, or using the revokeCertificate script.

4. • Query a certificate with queryCertificate(X509Certiifcate[], byte[], java.util.HashMap).

   ```
   public X509Certificate[] queryCertificate(byte[] certReq,
   java.util.HashMap customAttrs) throws WSPKIException;
   ```

This method is called by WebSphere Application Server runtime to query if certificate creation is completed on the CA. A X509Certificate[] is returned if certificate request is complete. A null is returned if the certificate request is pending.

You perform this operation from the administrative console using the Query (link to usec_sslperscertreqs.html) option, see “Personal certificate requests collection” on page 783and from a client using the queryCertificate script.

Results

the WSPKIClient interface for communicating with a certificate authority (CA) is implemented.

Creating a custom trust manager configuration for SSL

You can create a custom trust manager configuration at any management scope and associate the new trust manager with a Secure Sockets Layer (SSL) configuration.

Before you begin

You must develop, package, and locate a Java Archive JAR file for a custom key manager in the `was.install.root/lib/ext` directory on WebSphere Application Server. For more information, see “Example: Developing a custom trust manager for custom SSL trust decisions” on page 737.

About this task

Complete the following steps in the administrative console:

Procedure

1. Decide whether you want to create the custom trust manager at the cell scope or below the cell scope at the node, server, or cluster, for example.

   Important: When you create a custom trust manager at a level below the cell scope, you can associate it only with a Secure Sockets Layer (SSL) configuration at the same scope or higher. An SSL configuration at a scope lower than the trust manager does not see the trust manager configuration.

   - To create a custom trust manager at the cell scope, click **Security > SSL certificate and key management > Trust managers**. Every SSL configuration in the cell can select the trust manager at the cell scope.
   - To create a custom trust manager at a scope below the cell level, click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > *ssl_configuration* > Trust managers**.
2. Click **New** to create a new custom trust manager.
3. Type a unique trust manager name.
4. Select the **Custom** implementation setting. The custom setting enables you to define a Java class with an implementation of the javax.net.ssl.X509TrustManager Java interface and, optionally, the com.ibm.wsspi.ssl.TrustManagerExtendedInfo WebSphere Application Server interface.

   Note: The standard implementation setting applies only when the trust manager is already defined in the Java security provider list as a provider and an algorithm, which is not the case for a custom trust manager.
5. Type a class name, for example, com.ibm.test.CustomTrustManager.
6. Select one of the following actions:
   - Click **Apply**, then click **Custom properties** under Additional Properties to add custom properties to the new custom trust manager. When you are finished adding custom properties, click **OK** and **Save**, then go to the next step.

- Click **OK** and **Save**, then go to the next step.

7. Click **SSL certificate and key management** in the page navigation at the top of the panel.
8. Select one of the following actions:
   - Click **SSL configurations** under Related Items for a cell-scoped SSL configuration.
   - Click **Manage endpoint security configurations** to select an SSL configuration at a lower scope.
9. Click the link for the existing SSL configuration that you want to associate with the new custom trust manager. You can create a new SSL configuration instead of associating the custom trust manager with an existing configuration. For more information, see “Creating a Secure Sockets Layer configuration” on page 713.
10. Click **Trust and Key managers** under Additional Properties. If the new custom trust manager is not listed in the **Additional ordered trust managers** list, verify that you selected an SSL configuration scope that is at the same level or below the scope that you selected in Step 8.
11. Click **Add**. This action adds the new trust manager to the list of custom trust managers.
12. Click **OK** and **Save**.

Results

You have created a custom trust manager configuration that references a JAR file in the install directory of WebSphere Application Server and associates it with an SSL configuration during the connection handshake.

What to do next

You can create a custom trust manager for a pure client. For more information, see the TrustManagerCommands command group for the AdminTask object topic.

Trust and key managers settings

Use this page to specify trust and key managers for the selected SSL configuration.

To view this administrative console page, click **Security > SSL certificate and key management**. Under Configuration settings, click **Manage endpoint security configurations > {Inbound | Outbound} > *ssl_configuration***. Under **Related items** click **SSL configurations >** *SSL_configuration_name*. Under **Additional Properties** click **Trust and key managers**.

Attention: The application server checks the default trust managers first before checking the additional ordered trust managers in descending order.

Default trust manager:

Specifies the default trust manager. The default trust manager is IbmPKIX, which can be selected when certificate revocation checks must be made using the X509Certificate CRL distribution list. The other default trust manager is IbmX509.

Information	Value
Data type:	Text
Default:	ibmPKIX

Additional ordered trust managers:

Specifies additional trust managers that are used in the order shown for this SSL configuration.

Add:

Specifies to add the selection to the **Additional ordered trust managers** right-hand list.

Remove:

Specifies to remove the selection from the **Additional ordered trust managers** right-hand list.

Key manager:

Specifies the key manager that runs for this SSL configuration.

Information	Value
Data type:	Text
Default:	IbmX509

Trust managers collection

Use this page to define the implementation settings for the trust manager. A trust manager is a class that is invoked during a Secure Sockets Layer (SSL) handshake to make trust decisions about the remote end point. A default trust manager is used to validate the signature and expiration of the certificate. Custom trust managers can be plugged in to perform an extended certificate and host name check.

To view this administrative console page, click **Security > SSL certificate and key management**. Under Configuration settings, click **Manage endpoint security configurations > {Inbound | Outbound} > *ssl_configuration***. Under Related items, click **Trust managers**.

Table 96. Trust managers buttons. This table describes the trust manager buttons.

Button	Resulting action
New	Adds a new trust manager that can be selected by an SSL configuration. A trust manager is invoked during an SSL handshake and can decide whether the handshake should be accepted based on the information it knows about the remote certificate and host.
Delete	Deletes an existing trust manager. Make sure the trust manager is not referenced by any SSL configuration before you delete it.

Name:

Specifies the name of the trust manager. This name is used as a selection in the SSL configuration panel.

Class name:

Specifies a class that implements the javax.net.ssl.X509TrustManager interface. Optionally, the class can implement the com.ibm.wsspi.ssl.TrustMangerExtendedInfo interface to get extended information about the connection. The class can use the information to verify the host name and so on.

Algorithm:

Specifies the algorithm name of the trust manager that is implemented by the selected provider.

Trust managers settings

This page enables you to view and set definitions for trust manager implementation settings. A trust manager is a class that gets invoked during a Secure Sockets Layer (SSL) handshake to make trust decisions about the remote end point. A default trust manager is used to validate the signature and expiration of the certificate. Custom trust managers can be plugged in to perform an extended certificate and hostname check.

To view this administrative console page, click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > *ssl_configuration*** . Under Related items click **Trust managers > New** .

Name:

Specifies the name of the trust manager.

Information	Value
Data type:	Text
Default:	ibmX509TrustManager

Management scope:

Specifies the scope where this Secure Sockets Layer (SSL) configuration is visible. For example, if you choose a specific node, then the configuration is only visible on that node and any servers that are part of that node.

This field is not editable and provides information only.

Standard:

Specifies that the trust manager selection is available from a Java provider that is installed in the java.security file. This provider might be shipped by the Java Secure Sockets Extension (JSSE) or might be a custom provider that implements the javax.net.ssl.X509TrustManager interface.

Information	Value
Default:	Enabled

Provider:

Specifies the provider name that has an implementation of the javax.net.ssl.X509TrustManager interface. This provider is typically set to IBMJSSE2.

Enabled when **Standard** is selected.

Information	Value
Default	IBMJCE

Algorithm:

Specifies the algorithm name of the trust manager implemented by the selected provider.

Enabled when **Standard** is selected.

Information	Value
Default	ibmX509 or IbmPKIX
Range	ibmX509, IbmPKIX

Custom:

Specifies that the trust manager selection is based on a custom implementation class that implements the javax.net.ssl.X509TrustManager interface and optionally the com.ibm.wsspi.ssl.TrustManagerExctendedInfo interface to obtain additional connection information that is not otherwise available.

Information	Value
Default:	Disabled

Class name:

Specifies a class that implements the javax.net.ssl.X509TrustManager interface. Optionally, the class can implement the com.ibm.wsspi.ssl.TrustMangerExtendedInfo interface to get extended information about the connection. The class can use the information to verify the host name and so on.

Enabled when **Custom** is selected.

Information	Value
Data type:	Text

Example: Developing a custom trust manager for custom SSL trust decisions

The following example is of a sample custom trust manager. The custom trust manager makes no trust decisions but instead uses the information in the X.509 certificate that it references to make decisions.

After you build and package the custom trust manager, configure it either from the `ssl.client.props` file for a pure client or the SSLConfiguration TrustManager link in the administrative console. See "Trust manager control of X.509 certificate trust decisions" on page 677 for more information about trust managers.

Note: This example should only be used as a sample, and is not supported.

```
import java.security.cert.X509Certificate;
import javax.net.ssl.*;
import com.ibm.wsspi.ssl.TrustManagerExtendedInfo;

public final class CustomTrustManager implements X509TrustManager,
TrustManagerExtendedInfo
{
    private static ThreadLocal threadLocStorage = new ThreadLocal();
    private java.util.Properties sslConfig = null;
    private java.util.Properties props = null;

    public CustomTrustManager()
    {
    }

    /**
     * Method called by WebSphere Application Server run time to set the target
     * host information and potentially other connection info in the future.
     * This needs to be set on ThreadLocal since the same trust manager can be
     * used by multiple connections.
     *
     * @param java.util.Map - Contains information about the connection.
     */
    public void setExtendedInfo(java.util.Map info)
    {
        threadLocStorage.set(info);
    }

    /**
     * Method called internally to retrieve information about the connection.
     *
     * @return java.util.Map - Contains information about the connection.
     */
    private java.util.Map getExtendedInfo()
    {
        return (java.util.Map) threadLocStorage.get();
    }

    /**
     * Method called by WebSphere Application Server run time to set the custom
     * properties.
     *
     * @param java.util.Properties - custom props
     */
    public void setCustomProperties(java.util.Properties customProps)
    {
        props = customProps;
    }

    /**
     * Method called internally to the custom properties set in the Trust Manager
     * configuration.
     *
     * @return java.util.Properties - information set in the configuration.
     */
    private java.util.Properties getCustomProperties()
    {
        return props;
```

```
    }

    /**
     * Method called by WebSphere Application Server runtime to set the SSL
     * configuration properties being used for this connection.
     *
     * @param java.util.Properties - contains a property for the SSL configuration.
     */
    public void setSSLConfig(java.util.Properties config)
    {
        sslConfig = config;
    }

    /**
     * Method called by TrustManager to get access to the SSL configuration for
     * this connection.
     *
     * @return java.util.Properties
     */
    public java.util.Properties getSSLConfig ()
    {
        return sslConfig;
    }

    /**
     * Method called on the server-side for establishing trust with a client.
     * See API documentation for javax.net.ssl.X509TrustManager.
     */
    public void checkClientTrusted(X509Certificate[] chain, String authType)
        throws java.security.cert.CertificateException
    {
        for (int j=0; j<chain.length; j++)
        {
            System.out.println("Client certificate information:");
            System.out.println(  "Subject DN:"  + chain[j].getSubjectDN());
            System.out.println(  "Issuer DN:"  + chain[j].getIssuerDN());
            System.out.println(  "Serial number:"  + chain[j].getSerialNumber());
            System.out.println("");
        }
    }


    /**
     * Method called on the client-side for establishing trust with a server.
     * See API documentation for javax.net.ssl.X509TrustManager.
     */
    public void checkServerTrusted(X509Certificate[] chain, String authType)
        throws java.security.cert.CertificateException
    {
        for (int j=0; j<chain.length; j++)
        {
            System.out.println("Server certificate information:");
            System.out.println(  "Subject DN:"  + chain[j].getSubjectDN());
            System.out.println(  "Issuer DN:"  + chain[j].getIssuerDN());
            System.out.println(  "Serial number:"  + chain[j].getSerialNumber());
            System.out.println("");
        }
    }

    /**
     * Return an array of certificate authority certificates which are trusted
     * for authenticating peers. You can return null here since the IbmX509
     * or IbmPKIX will provide a default set of issuers.
     *
     * See API documentation for javax.net.ssl.X509TrustManager.
     */
    public X509Certificate[] getAcceptedIssuers()
    {
        return null;
    }
}
```

Creating a custom key manager for SSL

You can create a custom key manager configuration at any management scope and associate the new key manager with a Secure Sockets Layer (SSL) configuration.

Before you begin

You must develop, package, and locate a Java Archive (.JAR) file for a custom key manager in the `was.install.root/lib/ext` directory on WebSphere Application Server.

About this task

Complete the following steps in the administrative console:

Procedure

1. Decide whether you want to create the custom key manager at the cell scope or below the cell scope at the node, server, or cluster, for example.

   Important: When you create a custom key manager at a level below the cell scope, you can associate it only with a Secure Sockets Layer (SSL) configuration at the same scope or higher. An SSL configuration at a scope lower than the key manager does not see the key manager configuration.

   - To create a custom key manager at the cell scope, click **Security > SSL certificate and key management > Key managers**. Every SSL configuration in the cell can select the key manager at the cell scope.
   - To create a custom key manager at a scope below the cell level, click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > *SSL_configuration* > Key managers**.
2. Click **New** to create a new key manager.
3. Type a unique key manager name.
4. Select the **Custom** implementation setting. With the custom setting, you can define a Java class that has an implementation on the Java interface javax.net.ssl.X509KeyManager and, optionally, the com.ibm.wsspi.ssl.KeyManagerExtendedInfo WebSphere Application Server interface. The standard implementation setting applies only when the key manager is already defined in the Java security provider list as a provider and an algorithm, which is not the case for a custom key manager. The typical standard key manager is `algorithm = IbmX509, provider = IBMJSSE2`.
5. Type a class name. For example, com.ibm.test.CustomKeyManager.
6. Select one of the following actions:
   - Click **Apply**, then click **Custom properties** under Additional Properties to add custom properties to the new custom key manager. When you are finished adding custom properties, click **OK** and **Save**, then go to the next step.
   - Click **OK** and **Save**, then go to the next step.
7. Click **SSL certificate and key management** in the page navigation at the top of the panel.
8. Select one of the following actions:
   - Click **SSL configurations** under Related Items for a cell-scoped SSL configuration.
   - Click **Manage endpoint security configurations** to select an SSL configuration at a lower scope.
9. Click the link for the existing SSL configuration that you want to associate with the new custom key manager. You can create a new SSL configuration instead of associating the custom key manager with an existing configuration. For more information, see example below.
10. Click **Trust and Key managers** under Additional Properties.
11. Select the new custom key manager in the **Key manager** drop-down list. If the new custom key manager is not listed, verify that you selected an SSL configuration scope that is at the same level or below the scope that you selected in Step 8.
12. Click **OK** and **Save**.

Results

You have created a custom key manager configuration that references a JAR file in the installation directory of WebSphere Application Server and associates the custom configuration with an SSL configuration during the connection handshake.

Example

Developing a custom key manager for custom Secure Sockets Layer key selection. The following example is of a sample custom key manager. This simple key manager returns the configured alias if it is set using the alias properties com.ibm.ssl.keyStoreClientAlias or com.ibm.ssl.keyStoreServerAlias, depending on which side of the connection the key manager is used. The key manager defers to the JSSE default IbmX509 key manager to select an alias if these properties are not set.

After you build and package a custom key manager, you can configure it from either the `ssl.client.props` file for a pure client or by using the SSLConfiguration KeyManager link in the administrative console. See “Key manager control of X.509 certificate identities” on page 679 for more information about key managers.

Because only one key manager can be configured at a time for any given Secure Sockets Layer (SSL) configuration, the certificate selections on the server side might not work as they would when the default IbmX509 key manager is specified. When a custom key manager is configured, it is up to the owner of that key manager to ensure that the selection of the alias from the SSL configuration supplied is set properly when `chooseClientAlias` or `chooseServerAlias` are called. Look for the `com.ibm.ssl.keyStoreClientAlias` and `com.ibm.ssl.keyStoreServerAlias` SSL properties.

Note: This example should only be used as a sample, and is not supported.

```
package com.ibm.test;

import java.security.cert.X509Certificate;
import com.ibm.wsspi.ssl.KeyManagerExtendedInfo;

public final class CustomKeyManager
 implements javax.net.ssl.X509KeyManager, com.ibm.wsspi.ssl.KeyManagerExtendedInfo
{
    private java.util.Properties props = null;
    private java.security.KeyStore ks = null;
    private javax.net.ssl.X509KeyManager km = null;
    private java.util.Properties sslConfig = null;
    private String clientAlias = null;
    private String serverAlias = null;
    private int clientslotnum = 0;
    private int serverslotnum = 0;

    public CustomKeyManager()
    {
    }

    /**
     * Method called by WebSphere Application Server runtime to set the custom
     * properties.
     *
     * @param java.util.Properties - custom props
     */
    public void setCustomProperties(java.util.Properties customProps)
    {
        props = customProps;
    }

    private java.util.Properties getCustomProperties()
    {
        return props;
    }

    /**
     * Method called by WebSphere Application Server runtime to set the SSL
     * configuration properties being used for this connection.
     *
     * @param java.util.Properties - contains a property for the SSL configuration.
     */
    public void setSSLConfig(java.util.Properties config)
    {
        sslConfig = config;
    }

    private java.util.Properties getSSLConfig()
    {
        return sslConfig;
    }

    /**
     * Method called by WebSphere Application Server runtime to set the default
     * X509KeyManager created by the IbmX509 KeyManagerFactory using the KeyStore
```

```
     * information present in this SSL configuration.  This allows some delegation
     * to the default IbmX509 KeyManager to occur.
     *
     * @param javax.net.ssl.KeyManager defaultX509KeyManager - default key manager for IbmX509
     */
    public void setDefaultX509KeyManager(javax.net.ssl.X509KeyManager defaultX509KeyManager)
    {
        km = defaultX509KeyManager;
    }

    public javax.net.ssl.X509KeyManager getDefaultX509KeyManager()
    {
        return km;
    }

    /**
     * Method called by WebSphere Application Server runtime to set the SSL
     * KeyStore used for this connection.
     *
     * @param java.security.KeyStore - the KeyStore currently configured
     */
    public void setKeyStore(java.security.KeyStore keyStore)
    {
        ks = keyStore;
    }

    public java.security.KeyStore getKeyStore()
    {
        return ks;
    }

/**
     * Method called by custom code to set the server alias.
     *
     * @param String - the server alias to use
     */
    public void setKeyStoreServerAlias(String alias)
    {
        serverAlias = alias;
    }

    private String getKeyStoreServerAlias()
    {
        return serverAlias;
    }

    /**
     * Method called by custom code to set the client alias.
     *
     * @param String - the client alias to use
     */
    public void setKeyStoreClientAlias(String alias)
    {
        clientAlias = alias;
    }

    private String getKeyStoreClientAlias()
    {
        return clientAlias;
    }

    /**
     * Method called by custom code to set the client alias and slot (if necessary).
     *
     * @param String - the client alias to use
     * @param int - the slot to use (for hardware)
     */
    public void setClientAlias(String alias, int slotnum) throws Exception
    {
        if ( !ks.containsAlias(alias))
        {
            throw new IllegalArgumentException ( "Client alias " + alias + "
            not found in keystore." );
        }
        this.clientAlias = alias;
        this.clientslotnum = slotnum;
    }

    /**
     * Method called by custom code to set the server alias and slot (if necessary).
     *
     * @param String - the server alias to use
     * @param int - the slot to use (for hardware)
     */
    public void setServerAlias(String alias, int slotnum) throws Exception
    {
        if ( ! ks.containsAlias(alias))
        {
            throw new IllegalArgumentException ( "Server alias " + alias + "
            not found in keystore." );
```

```
    }
    this.serverAlias = alias;
    this.serverslotnum = slotnum;
}


/**
 * Method called by JSSE runtime to when an alias is needed for a client
 * connection where a client certificate is required.
 *
 * @param String keyType
 * @param Principal[] issuers
 * @param java.net.Socket socket (not always present)
 */
public String chooseClientAlias(String[] keyType, java.security.Principal[]
issuers, java.net.Socket socket)
{
    if (clientAlias != null && !clientAlias.equals(""))
    {
        String[] list = km.getClientAliases(keyType[0], issuers);
        String aliases = "";

        if (list != null)
        {
            boolean found=false;
            for (int i=0; i<list.length; i++)
            {
                aliases += list[i] + " ";
                if (clientAlias.equalsIgnoreCase(list[i]))
                    found=true;
            }

            if (found)
            {
                return clientAlias;
            }

        }
    }

    // client alias not found, let the default key manager choose.
    String[] keyArray = new String [] {keyType[0]};
    String alias = km.chooseClientAlias(keyArray, issuers, null);
    return alias.toLowerCase();
}

/**
 * Method called by JSSE runtime to when an alias is needed for a server
 * connection to provide the server identity.
 *
 * @param String[] keyType
 * @param Principal[] issuers
 * @param java.net.Socket socket (not always present)
 */
public String chooseServerAlias(String keyType, java.security.Principal[]
issuers, java.net.Socket socket)
{
    if (serverAlias != null && !serverAlias.equals(""))
    {
        // get the list of aliases in the keystore from the default key manager
        String[] list = km.getServerAliases(keyType, issuers);
        String aliases = "";

        if (list != null)
        {
            boolean found=false;
            for (int i=0; i<list.length; i++)
            {
                aliases += list[i] + " ";
                if (serverAlias.equalsIgnoreCase(list[i]))
                    found = true;
            }

            if (found)
            {
                return serverAlias;
            }
        }
    }

    // specified alias not found, let the default key manager choose.
    String alias = km.chooseServerAlias(keyType, issuers, null);
    return alias.toLowerCase();
}

public String[] getClientAliases(String keyType, java.security.Principal[] issuers)
{
    return km.getClientAliases(keyType, issuers);
}
```

```
    public String[] getServerAliases(String keyType, java.security.Principal[] issuers)
    {
        return km.getServerAliases(keyType, issuers);
    }

    public java.security.PrivateKey getPrivateKey(String s)
    {
        return km.getPrivateKey(s);
    }

    public java.security.cert.X509Certificate[] getCertificateChain(String s)
    {
        return km.getCertificateChain(s);
    }

    public javax.net.ssl.X509KeyManager getX509KeyManager()
    {
        return km;
    }

}
```

What to do next

You can create a custom key manager for a pure client. For more information, see the keyManagerCommands command group for the AdminTask object.

Associating a Secure Sockets Layer configuration dynamically with an outbound protocol and remote secure endpoint

After you create a Secure Sockets Layer (SSL) configuration, you must associate a secure outbound management scope with the new configuration. In this release, you can associate one SSL configuration with one remote secure endpoint and a different SSL configuration to another remote secure endpoint. Both endpoints can use the same outbound protocol, if appropriate. This task describes how to create the association dynamically.

Before you begin

Dynamic outbound selection requires that you provide only the outbound protocol name, the target host, and the target port so that WebSphere Application Server can make a connection between the SSL configuration and the outbound protocol or remote secure endpoint. The dynamic outbound selection method takes precedence over other selection methods, such as central management and direct selection, but is second to the programmatic method, that is, setting an SSL configuration on the running thread. For more information about the selection types and precedence rules, see “Secure communications using Secure Sockets Layer (SSL)” on page 667.

About this task

Complete the following steps in the administrative console:

Procedure

1. Click **Security > SSL certificate and key management > Manage endpoint security configurations > Outbound**.
2. Select the management scope that you want to associate with an SSL configuration on the topology tree.
3. Under Related Items, click **Dynamic outbound endpoint SSL configurations**. The default dynamic outbound configuration name, the target protocol, host, and port connection information, and the SSL configuration name display.
4. Click **New** to create a new dynamic outbound configuration.
5. Type a dynamic outbound configuration name. Use a name that is descriptive of the purpose of the dynamic selection configuration.
6. Optionally, type a dynamic selection configuration description.

7. Type the connection information that you want to associate with the configuration that is displayed in the SSL configuration drop-down list. The connection information must be in the format *protocol name*, *target host*, *target port*. You can substitute an asterisk (*) for any value, as in the following examples, where 443 is a port, www.mycompany.com is a host, HTTP is a protocol, and .hometown.mycompany.com is a target host. You can add multiple connections, but each additional connection can affect outbound performance.
   - ***,*,443**
   - ***,www.mycompany.com,443**
   - **HTTP,.hometown.mycompany.com,***
   - ***,*,***

     gotcha: Do not use this configuration because it matches all outbound specifications. Therefore, no other SSL configuration is used for outbound connections.

   gotcha:
   - Unless the intention is to set the protocol property through the JSSEHelper API, the protocol filter should be set to * (as in the first two examples). See "Dynamic Selection" in “Secure communications using Secure Sockets Layer (SSL)” on page 667 for more information.
   - The connection protocols that are used for dynamic outboud SSL configuration selection, that are illustrated in the preceding examples, which are not corresponding the protocol name of the URL. To use one of these protocols from a user-written application, programmatic SSL configuration selection must be implemented.
8. Click **Add** to add the new connection to the set of SSL configuration connections. To remove a connection, select it and click **Remove**.
9. Select an SSL configuration from the list.
10. Click **Get certificate aliases** to refresh the certificate aliases that are contained in the associated key store.
11. Choose a certificate alias from the list.
12. Click **OK** and **Save**.

Results

WebSphere Application Server is ready to connect one or more SSL configurations to one or more remote secure endpoints.

What to do next

You can return to the outbound tree and select another management scope to associate with the same or a new outbound configuration.

Programmatically specifying an outbound SSL configuration using JSSEHelper API

WebSphere Application Server provides a way to specify programmatically which Secure Sockets Layer (SSL) configurations to use prior to making an outbound connection. The com.ibm.websphere.ssl.JSSEHelper interface provides a complete set of application programming interfaces (APIs) for handling SSL configurations.

About this task

Perform the following steps for your application when using the JSSEHelper API to establish an SSL properties object on the thread for use by the runtime. Some of these APIs have Java 2 Security permission requirements. See the JSSEHelper API documentation for more information about the permissions required by your application.

Select the approach that best fits your connection situation when you specify programmatically which Secure Sockets Layer (SSL) configurations to use prior to making an outbound connection.

Procedure

1. Obtain an instance of the JSSEHelper API.

```
com.ibm.websphere.ssl.JSSEHelper jsseHelper  = com.ibm.websphere.ssl.JSSEHelper.getInstance();
```

2. Obtain SSL properties from the WebSphere Application Server configuration or use those provided by your application. Use one of the following options.
   - By direction selection of an alias name, within the same management scope or higher as in the following example:

```
try
{  String alias = "NodeAServer1SSLSettings";
  // As specified in the WebSphere SSL configuration  Properties
 sslProps = jsseHelper.getProperties(alias); }
 catch (com.ibm.websphere.ssl.SSLException e)
 {  e.printStackTrace();   // handle exception }
```

   - By using the getProperties API for programmatic, direction, dynamic outbound, or management scope selection (based on precedence rules and inheritance). The SSL runtime uses the getProperties API to determine which SSL configuration to use for a particular protocol. This decision is based on both the input (sslAlias and connectionInfo) and the management scope from which the property is called. The getProperties API makes decisions in the following order:
     a. The API checks the thread to see if properties already exist.
     b. The API checks for a dynamic outbound configuration that matches the ENDPOINT_NAME, REMOTE_HOST, and or REMOTE_PORT.
     c. The API checks to see if the optional sslAlias property is specified. You can configure any protocol as direct or centrally managed. When a protocol is configured as direct, the sslAlias parameter is `null`. When a protocol is configured as centrally managed, the sslAlias parameter is also `null`.
     d. If no selection has been made, the API chooses the dynamic outbound configuration based on the management scope it was called from. If the dynamic outbound configuration is not defined in the same scope, it then searches the hierarchy to locate one.

     The last choice is the cell-scoped SSL configuration (in WebSphere Application Server, Network Deployment) or the node-scoped SSL configuration (in Base Application Server). The com.ibm.websphere.ssl.SSLConfigChangeListener parameter is notified when the SSL configuration that is chosen by a call to the getProperties API changes. The protocol can then call the API again to obtain the new properties as in the following example:

```
try {  String sslAlias = null;
 // The sslAlias is not specified directly at this time.  String host = "myhost.austin.ibm.com";
 // the target host  String port = "443";
 // the target port   HashMap connectionInfo = new HashMap();
 connectionInfo.put(JSSEHelper.CONNECTION_INFO_DIRECTION,    JSSEHelper.DIRECTION_OUTBOUND);
 connectionInfo.put(JSSEHelper.CONNECTION_INFO_REMOTE_HOST, host);
 connectionInfo.put(JSSEHelper.CONNECTION_INFO_REMOTE_PORT,    Integer.toString(port));
 connectionInfo.put(JSSEHelper.CONNECTION_INFO_ENDPOINT_NAME,    JSSEHelper.ENDPOINT_IIOP);
             java.util.Properties props = jsseHelper.getProperties(sslAlias,    connectionInfo, null); }
 catch (com.ibm.websphere.ssl.SSLException e)
 {  e.printStackTrace();   // handle exception }
```

   - By creating your own SSL properties and then passing them to the runtime, as in the following example:

```
try {
 // This is the recommended "minimum" set of SSL properties. The trustStore can
  // be the same as the keyStore.  Properties sslProps = new Properties();
 sslProps.setProperty("com.ibm.ssl.trustStore", "some value");
 sslProps.setProperty("com.ibm.ssl.trustStorePassword", "some value");
```

```
 sslProps.setProperty("com.ibm.ssl.trustStoreType", "some value");
 sslProps.setProperty("com.ibm.ssl.keyStore", "some value");
 sslProps.setProperty("com.ibm.ssl.keyStorePassword", "some value");
 sslProps.setProperty("com.ibm.ssl.keyStoreType", "some value");
  jsseHelper.setSSLPropertiesOnThread(sslProps); }
 catch (com.ibm.websphere.ssl.SSLException e)
{  e.printStackTrace();   // handle exception }
```

3. Use the JSSEHelper.setSSLPropertiesOnThread(props) API to set the Properties object on the thread so that the runtime picks it up and uses the same JSSEHelper.getProperties API. You can also obtain properties from the thread after they are set with the jsseHelper.getSSLPropertiesOnThread() API, as in the following example:

```
try
{  Properties sslProps = jsseHelper.getProperties(null,    connectionInfo, null);
jsseHelper.setSSLPropertiesOnThread(sslProps); }
 catch (com.ibm.websphere.ssl.SSLException e)
{  e.printStackTrace();   // handle exception }
```

4. When the connection is completed, you must clear the SSL properties from the thread by passing the `null` value to the setPropertiesOnThread API.

```
try
 {  jsseHelper.setSSLPropertiesOnThread(null); }
 catch (com.ibm.websphere.ssl.SSLException e)
 {  e.printStackTrace();   // handle exception }
```

Associating Secure Sockets Layer configurations centrally with inbound and outbound scopes

After you create a Secure Sockets Layer (SSL) configuration, you must associate a secure inbound or outbound management scope with the new configuration. You can manage the association centrally so that you can easily make changes that affect all the scopes that are lower on the topology and that are associated with the configuration. Beginning with WebSphere Application Server version 6.1, the recommended and the default configuration method is centrally managed SSL configurations.

Before you begin

You can simplify the number of associations that you need to make for an SSL configuration by associating the configuration with the highest level management scope requiring a unique configuration. SSL configuration associations manifest inheritance behaviors. Because of the inheritance behaviors, all of the scopes that are lower on the topology inherit this SSL configuration. For example, an association you make at the cell level affects nodes, servers, clusters, and endpoints. For more information, see “Central management of SSL configurations” on page 686.

A precedence rule determines which SSL configuration association is used at a particular scope. The highest precedence is given to endpoints on the topology. If you establish an association at the endpoint, this association overrides any prior association that you made higher up on the management scope topology.

About this task

Complete the following steps in the administrative console:

Procedure

1. Click **Security > SSL certificate and key management**.
2. Select the **Dynamically update the runtime when SSL configuration changes** check box if you want changes that you make to an existing SSL configuration to occur dynamically. All outbound SSL communications honor the dynamic SSL changes. Protocols that do not use the channel frameworks SSL channel for inbound communications, including Object Request Broker (ORB) and administrative SOAP protocols, do not honor dynamic updates. For more information, see “Dynamic configuration updates in SSL” on page 705.
3. Click **Manage endpoint security configurations**.
4. Select either the inbound or the outbound tree. After finishing the selected tree, you can return to this step to repeat the following steps for the other tree.

5. Click the link for the selected cell, node, node group, server, cluster, or endpoint on the topology tree. If the scope already has an associated SSL configuration and alias, these objects display in parentheses immediately following the scope name, for example: Node01(NodeDefaultSSLSettings,default). If the deployment manager has federated a node, the node scope SSL configuration overrides the cell scope configuration above it in the topology.
6. Decide whether to override the inherited values that display in the read-only fields. Read-only fields include the management scope name, the direction, and the inherited SSL configuration name and certificate alias.
   - If you are satisfied with these values, do not override them.
   - If you want to override the inherited values, select the **Override inherited values** check box.
7. Select an SSL configuration from the list.
8. Click **Update certificate alias list**. The certificate alias list comes from the key store that is referenced by the new SSL configuration.
9. Click **Manage certificates** if you want to manage the personal certificates that are contained in the key store that is referenced in the SSL configuration.
10. Click **Update certificate alias list** to refresh the list of aliases.
11. Select a certificate alias in the key store to represent the identity of the endpoint.
12. Click **OK** to save your changes.
13. Click **Manage endpoint security configurations and trust zones** to return to the topology tree.
14. Configure the opposite direction on the topology tree using the steps in this task. You can also select additional scopes to associate with the SSL configuration, as needed.

Results

Each SSL configuration at the selected scope and at scopes beneath it on the topology tree have the same SSL configuration properties. The following SSL configuration methods override the centrally managed configurations that you associate in the tree view:

- Direct selection at the endpoint
- Dynamic outbound SSL configuration associations
- Programmatic specifications

What to do next

At any management scope, you can configure the following objects: dynamic outbound endpoint SSL configurations, key stores, key sets, key set groups, key managers, and trust managers. Like SSL configurations, these objects are scoped automatically so that they are not visible higher up in the tree nor are they loaded during runtime by processes that are higher up in the tree.

Selecting an SSL configuration alias directly from an endpoint configuration

You can associate a secure outbound endpoint with a new Secure Sockets Layer (SSL) configuration directly. If you are migrating from a release prior to version 6.1, WebSphere Application Server still supports configurations that were selected directly at an endpoint. Direct selection always overrides centrally managed configurations and preserves migrated configurations.

About this task

Select an SSL configuration alias directly at the following endpoints:

- **Security > Global security > RMI/IIOP security > CSIv2 outbound transport**
- **Security > Global security > RMI/IIOP security > CSIv2 inbound transport**
- **System administration > Deployment manager > Transport Chain > WCInboundAdminSecure > SSL inbound channel (SSL_1)**

- **System administration > Deployment manager > Administration Services > JMX connectors > SOAPConnector > Custom Properties > sslConfig**
- **System administration > Node agents > nodeagent > Administration Services > JMX connectors > SOAPConnector > Custom Properties > sslConfig**
- **Servers > Application servers > server1 > Messaging engine inbound transports > InboundSecureMessaging > SSL inbound channel (SIB_SSL_JFAP)**
- **Servers > Application servers > server1 > WebSphere MQ link inbound transports > InboundSecureMQLink > SSL inbound channel (SIB_SSL_MQFAP)**
- **Servers > Application servers > server1 > SIP Container Settings > SIP container transport chains > SIPCInboundDefaultSecure > SSL inbound channel (SSL_5)**
- **Servers > Application servers > server1 > Web Container Settings > Web container transport chains > WCInboundAdminSecure > SSL inbound channel (SSL_1)**
- **Servers > Application servers > server1 > Web Container Settings > Web container transport chains > WCInboundDefaultSecure > SSL inbound channel (SSL_2)**

Attention: The central management of SSL configurations can be a more efficient strategy because multiple configurations can be contained within a single SSLConfigGroup. If you need to convert configuration references that are already directly managed to centrally managed configurations, modify each endpoint individually. Use the **AdminConfig.modify** command to set the `sslConfigAlias` value to an empty string (""). Below is an example of doing this:

- Using Jacl:

```
set s1 [$AdminConfig getid /Cell:mycell/Node:mynode/Server:server1/]
set sslChannel [lindex [$AdminConfig list SSLInboundChannel $s1] 0]
$AdminConfig modify $sslChannel [list[list sslConfigAlias ""]]
```

For more information on using this command, see the information about configuring processes using scripting.

For more information on specific wsadmin commands that affect a repertoire as opposed to individual endpoints, see the SSLConfigGroupCommands group for the AdminTask topic.

Complete the following steps in the administrative console:

Note: These steps provide an example to follow when you directly select any of the endpoints listed above.

Procedure

1. Click **Security > Global security > RMI/IIOP security > CSIv2 outbound transport**.
2. Click **Use specific SSL alias**. When you identify a specific SSL alias, you override the centrally managed scope associations.
3. Select an SSL configuration alias from the drop-down list.
4. Click **OK**.
5. Repeat these steps for additional protocols or endpoints, if desired.

Results

By associating the endpoint directly, you have overridden a centrally managed SSL configuration.

What to do next

If you decide to use management scopes instead of endpoints to associate an SSL configuration, follow the steps above, but click **Centrally managed** instead of **Use specific SSL alias**, then click **Manage endpoint security configurations**. The console is redirected to **Security > SSL certificate and key management > Manage endpoint security configurations**.

Enabling Secure Sockets Layer client authentication for a specific inbound endpoint

When you establish a Secure Sockets Layer (SSL) configuration, you can enable client authentication for a specific inbound endpoint.

Before you begin

The endpoint configuration must already exist in the SSL topology.

About this task

Complete the following steps in the administrative console:

Procedure

1. Click **Security > SSL certificate and key management > Manage endpoint security configurations > Inbound >** *SSL_configuration*. If you want to enable SSL client authentication for all processes, define an SSL configuration for the new endpoint at the node or cell level so that it is visible to all processes on the same node or on the entire cell. For more information, see "Creating a Secure Sockets Layer configuration" on page 713.
2. Select **Override inherited values**. The SSL configuration is used for the current scope and any lower scopes that have not already designated an SSL configuration. This field displays for server and node groups within the object hierarchy and does not display for the top-level node or cell.
3. Select an SSL configuration from the drop-down list.
4. Click **Update certificate alias list**.
5. Select a **Certificate alias** from the drop-down list.
6. Click **OK** to save the configuration.

Results

You can repeat the previous steps for each endpoint that uses the same SSL configuration to enable client authentication for the inbound endpoints.

What to do next

CSIv2 Protocol Exception:

The Common Secure Interoperability Version 2 (CSIv2) secure endpoints, used for Remote Method Invocation over the Internet Inter-ORB Protocol (RMI/IIOP) security, cannot override inherited values. While the rest of the SSL properties are effective for CSIv2 when they are selected at the centrally-managed Secure Communications panel, the client authentication selection is controlled by the CSIv2 protocol configuration.

To enable SSL client certificate authentication for the CSIv2 protocol, you must use the CSIv2 inbound and outbound authentication panels. For SSL client authentication to occur between two servers, you must enable (support or require) SSL client certificate authentication for both the inbound and the outbound policies.

WebSphere Application Server can either request (support) clients to provide signer certificates for the SSL handshake, or the server can require clients to provide a valid signer certificate for the SSL handshake, which is a more secure method. However, when the server requires certificates, the server must obtain a signer for each client that connects to the server, which involves more server-side management.

The client certificate should not be used for the identity when it is used from server-to-server. However, when a pure client sends the client certificate it is used for the identity unless a message level identity is specified, such as a user ID or a password.

Do the following to enable client certificate authentication for the CSIv2 protocol for server-to-server:

1. **Click Security > Global security**.
2. Expand the **RMI/IIOP** security section.
3. Click **CSIv2 inbound authentication**.
4. Under Client authentication, select either **supported** or **required**. When you select required, only one SSL port is opened (CSV2_SSL_MUTUALAUTH_LISTENER_ADDRESS). When you select supported, two SSL ports are opened (both CSIV2_SSL_MUTUALAUTH_LISTENER_ADDRESS and CSIV2_SSL_SERVERAUTH_LISTENER_ADDRESS).

   If there are two ports, the client can select either based on the security configuration policy of the port.
5. Click **OK** to save.
6. If you want server-to-server SSL client authentication, then complete the remaining steps. If you don't complete the remaining steps, only pure clients are enabled to send client certificates.
7. Expand the **RMI/IIOP** security section.
8. Click **CSIv2 outbound authentication**.
9. Under Client authentication, select either **supported** or **required**.

The SSL configuration for the inbound secure endpoints for which you enable SSL client certificate authentication must have the signer certificate from any client that attempts to open a connection to that inbound secure endpoint. You must collect those signers and then add them to the trust store associated with the inbound secure endpoints SSL configuration.

Manage endpoint security configurations

Use this page to select a Secure Socket Layer (SSL) configuration from the Local Topology hierarchy, which includes cells, nodes, node groups, servers, and clusters.

To view this administrative console page, click **Security > SSL certificate and key management**. Under Configuration settings, click **Manage endpoint security configurations**.

Local topology:

The Local topology represents the hierarchy of nodes, node groups, clusters, servers, and end points within the cell that comprise a centralized SSL configuration.

The topology acts as a hierarchical tree in terms of inheritance. For example, if an SSL configuration has been associated with a specific node, then all servers within that node will inherit that SSL configuration selection, provided the servers are not associated with an SSL configuration at the server scope. Centralized management of SSL is the default configuration; however, it can be overridden at various locations to directly select a specific SSL alias as in previous releases for backwards compatibility.

Scope	Description
Inbound/Outbound	Specifies the topology tree in terms of connection direction. For example, the inbound tree represents all server endpoints that receive connections at the various servers within the cell. The outbound tree represents the client side of connections from the various servers within the cell.
Nodes	Specifies the nodes that are part of the cell. The list of nodes is updated anytime a node gets federated into the cell.

Scope	Description
Servers	Specifies the servers that are part of a specific node. You can enable a specific server to have an SSL configuration associated with it so that resources within the same server can use the associated SSL configuration.
Clusters	Specifies the clusters that are part of the cell. When an SSL configuration is associated with a cluster, all servers within the cluster will use the same SSL configuration unless specified at a lower level in the topology.
Nodegroups	Specifies the node groups that are part of the cell. When an SSL configuration is associated with a node group, all nodes within that node group may use the same SSL configuration unless one is specified at a lower scope in the topology or the specific end point has chosen a direct alias reference.
Secure port and transport	Specifies an endpoint name to associate with an SSL configuration when more specific SSL settings are needed at this level. You could select an alias directly at the endpoint panel; however, when you use **Secure port and transport**, you can maintain more centralized control of the SSL configuration and make changes more easily.

Dynamic inbound and outbound endpoint SSL configurations collection

Use this page to manage dynamic endpoint Secure Sockets Layer (SSL) configurations, which represent associations between Secure Socket Layer (SSL) configurations and their target protocol, host, and port.

To view this administrative console page, click **Security > SSL certificate and key management**. Under Configuration settings, click **Manage endpoint security configurations > {Inbound | Outbound} > *ssl_configuration***. Under Related items, click **Dynamic outbound endpoint SSL configurations**.

When an outbound connection is attempted, this association is checked ahead of the SSL configuration scope association. Based on the target protocol,host,port, the outbound SSL configuration used can be different from the default specified in the SSL scope configuration.

Table 97. Dynamic inbound and outbound endpoint SSL configurations buttons. This table lists the dynamic inbound and outbound endpoint SSL configuration buttons.

Button	Resulting action
New	Adds a new dynamic outbound selection criteria. The outbound connection selects an SSL configuration based upon connection information, including DNS host name and domain, port, and protocol type. When an outbound connection is being made, the dynamic outbound selection criteria are queried for a match, and if found the SSL configuration associated is used.
Delete	Deletes an existing dynamic outbound endpoint SSL configuration.

Name:

Specifies the unique name of the dynamic endpoint configuration.

Connection information:

Specifies the set of target protocol, host, port for the outbound request in the form *protocol*,*host*,*port*.

SSL Configuration:

Specifies the SSL configuration that is used by requests at this scope when a match occurs for the given selection criteria.

Dynamic outbound endpoint SSL configuration settings

Use this page to set properties for dynamic outbound endpoint SSL configurations, which represent associations between SSL configurations and their target protocol, host, and port.

To view this administrative console page, click **Security > SSL certificate and key management**. Under Configuration settings, click **Manage endpoint security configurations > {Inbound | Outbound} > *ssl_configuration***. Under Related items, click **Dynamic [inbound | outbound] endpoint SSL configurations**. Then click the **New** button.

When an outbound connection is attempted, this association is checked ahead of the Secure Sockets Layer (SSL) configuration scope association. This means based on the target protocol,host,port, the outbound SSL configuration used can be different than the default specified in the SSL scope configuration.

Name:

Specifies the unique name of the dynamic endpoint configuration.

Information	Value
Data type:	Text

Description:

Specifies text that describes the purpose of this dynamic selection criteria.

Information	Value
Data type:	Text

Add connection information:

Specifies select information in the form protocol,host,port for the outbound connection. Multiple selection criteria can be entered. All of the connection information for dynamic outbound selection might not be available, and you may have to adjust the dynamic outbound selection connection filter and fill in an asterisk (*) for the missing part of the connection information. An asterisk (*) can be used to mean all protocols, hosts, or ports. You can use an asterisk(*) for any field.

Information	Value
Data type:	Text

An example of selection criteria is `*,www.ibm.com,*`, which means that any time the target host is www.ibm.com, you must use the SSL configuration specified here. Another example selection criteria is `IIOP,*,*`, which means that any outbound IIOP request uses the SSL configuration that is specified in the SSL configuration field. When there is a conflict between two selection criteria, the application server uses the first match. The list of valid protocols you can use include: IIOP, HTTP, JMS, LDAP, SIP, ADMIN_SOAP, ADMIN_IIOP, or WEBSERVICES_HTTP.

When user written applications are expecting to take advantage of dynamic outbound selections, know that not all connection information may be available. For example, the openConnection() call on an URL object ultimately calls `createSocket(java.net.Socket socket, String host, int port, boolean autoClose)`. The connection information can be built with the host and port provided, but there is no protocol provided. In this case, a wild card, an asterisk (*), should be used for the protocol part of the dynamic selection connection information.

Add:

Specifies to add the selected information from the **Add select information** menu to the right-hand list.

Remove:

Specifies to remove the selection from the right-hand list.

SSL Configuration:

Specifies the SSL configuration to be used by requests at this scope when a match occurs for the given selection criteria.

Information	Value
Data type:	Text

Get certificate alias:

When selected, the keystore within the selected SSL configuration is queried for a list of personal certificates from which to choose.

Certificate alias:

Specifies the certificate alias that is used as the identity for the connection.

If you select **None**, the Java Secure Sockets Extension (JSSE) key manager determines which certificate is used. If multiple certificates exist in the keystore, the key manager might not consistently select the same certificate.

Information	Value
Data type:	Text
Default:	(none)

Quality of protection (QoP) settings

Use this page to specify security level, ciphers, and mutual authentication settings for the Secure Socket Layer (SSL) configuration.

To view this administrative console page, click **Security > SSL certificate and key management**. Under Configuration settings, click **Manage endpoint security configurations > {Inbound | Outbound} > *ssl_configuration***. Under Related items, click **SSL configurations >** . Click on {*SSL_configuration_name* }. Under **Additional Properties**, click **Quality of protection (QoP) settings**.

Client authentication

Specifies the whether SSL client authentication should be requested if the SSL connection is used for the server side of the connection.

If `None` is selected, the server does not request that a client certificate be sent during the handshake. If `Supported` is selected, the server requests that a client certificate be sent. If the client does not have a certificate, the handshake might still succeed. If `Required` is selected, the server requests that a client certificate be sent. If the client does not have a certificate, the handshake fails.

Information	Value
Data type:	Text
Default:	None

Protocol

Specifies the Secure Sockets Layer (SSL) handshake protocol. This protocol is typically SSL_TLS, which supports all handshake protocols except for SSLv2 on the server side. When United States Federal Information Processing standard (FIPS) option is enabled, Transport Layer Security (TLS) is automatically used regardless of this setting.

Information	Value
Data type:	text
Default:	SSL_TLS

Predefined JSSE provider

Specifies one of the predefined Java Secure Sockets Extension (JSSE) providers. The IBMJSSE2 provider is recommended for use on all platforms which support it. It is required for use by the channel framework SSL channel. When Federal Information Processing Standard (FIPS) is enabled, IBMJSSE2 is used in combination with the IBMJCEFIPS crypto provider.

Information	Value
Default:	Enabled

Select provider

Specifies a package that implements a subset of the cryptography aspects for the Java security application programming interface (API). This value is a JSSE provider name that is listed in the java.security file. Note that cipher suites and protocol values depend upon the provider.

Information	Value
Data type:	Text
Default:	IBMJSSE2

Custom JSSE provider

Specifies that a custom JSSE provider should be used.

Information	Value
Default:	Disabled

Custom provider

Specifies a package that implements a subset of the cryptography aspects for the Java security application programming interface (API). This value is a Java Secure Sockets Extension (JSSE) provider name that is listed in the java.security file. Note that cipher suites and protocol values depend upon the provider.

Information	Value
Data type:	Text

Cipher suite groups

Specifies the various cipher suite groups that can be chosen depending upon your security needs. The stronger the cipher suite strength, the better the security; however, this can result in performance consequences.

Information	Value
Data type:	Text
Default:	Strong

Update selected ciphers
When selected, the cipher suites that are contained within the selected Cipher suite group are added to the list of **Selected ciphers**. Any change to this list changes the Cipher suite group to `custom`.

Selected ciphers
Specifies the ciphers that are effective when the configuration is saved. These ciphers are used to negotiate with the remote side of the connection during the handshake. A common cipher needs to be selected or the handshake fails.

Information	Value
Data type:	Text

Add
Specifies to add the selected cipher to the **Selected ciphers** list.

Remove
Specifies to remove the selected cipher from the **Selected ciphers** list.

ssl.client.props client configuration file
Use the `ssl.client.props` file to configure Secure Sockets Layer (SSL) for clients. In previous releases of WebSphere Application Server, SSL properties were specified in the `sas.client.props` or `soap.client.props` files or as system properties. By consolidating the configurations, WebSphere Application Server enables you to manage security in a manner that is comparable to server-side configuration management. You can configure the `ssl.client.props` file with multiple SSL configurations.

Setting up the SSL configuration for clients

Client runtimes are dependent on the WebSphere Application Server `ssl.client.props` configurations.

Use the `setupCmdLine.bat` script on the command line to specify the com.ibm.SSL.ConfigURL system property.

The com.ibm.SSL.ConfigURL property references a file URL that points to the `ssl.client.props` file. You can reference the CLIENTSSL variable on the command line of any script that uses the `setupCmdLine.bat` file.

When you specify the com.ibm.SSL.ConfigURL system property, the SSL configuration is available to all protocols that use SSL. SSL configurations, which are referenced in the `ssl.client.props` file, also have aliases that you can reference. In the following sample code from the `sas.client.props` file, all of the SSL properties are replaced with a property that points to an SSL configuration in the `ssl.client.props` file:

```
com.ibm.ssl.alias=DefaultSSLSettings
```

The following sample code shows a property in the `soap.client.props` file that is similar to the com.ibm.SSL.ConfigURL property. This property references a different SSL configuration on the client side:

```
com.ibm.ssl.alias=DefaultSSLSettings
```

In the `ssl.client.props` file, you can change the administrative SSL configuration to avoid modifying the `soap.client.props` file.

Tip: Support for SSL properties is still specified in the `sas.client.props` and `soap.client.props` files. However, consider moving the SSL configurations to the `ssl.client.props` file, because this file is the new configuration model for client SSL.

Properties of the ssl.client.props file

This section describes the default `ssl.client.props` file properties in detail, by sections within the file. Be aware that if you specify javax.net.ssl system properties, these will override the settings in `ssl.client.props` file.

Global properties:

Global SSL properties are process-specific properties that include Federal Information Processing Standard (FIPS) enablement, the default SSL alias, the user.root property for specifying the root location of the key and truststore paths, and so on.

Table 98. Properties of the ssl.client.props file. This table describes the properties of the `ssl.client.props` file.

Property	Default	Description
com.ibm.ssl.defaultAlias	DefaultSSLSettings	Specifies the default alias that is used whenever an alias is not specified by the protocol that calls the JSSEHelper API to retrieve an SSL configuration. This property is the final arbiter on the client side for determining which SSL configuration to use.
com.ibm.ssl.validationEnabled	false	When set to `true`, this property validates each SSL configuration as it is loaded. Use this property for debug purposes only, to avoid unnecessary performance overhead during production.
com.ibm.ssl.performURLHostNameVerification	false	When set to `true`, this property enforces URL host name verification. When HTTP URL connections are made to target servers, the common name (CN) from the server certificate must match the target host name. Without a match, the host name verifier rejects the connection. The default value of `false` omits this check. As a global property, it sets the default host name verifier. Any javax.net.ssl.HttpsURLConnection object can choose to enable host name verification for that specific instance by calling the setHostnameVerifier method with its own HostnameVerifier instance. **gotcha:** This property does not apply to SSL channels.
com.ibm.security.useFIPS	false	When set to `true`, FIPS-compliant algorithms are used for SSL and other Java Cryptography Extension (JCE)-specific applications. This property is typically not enabled unless the property is required by the operating environment.
com.ibm.websphere.security.FIPSLevel	false	Specifies the level of the security standard to use. Valid values include 140-2, SP800-131 and transition. The com.ibm.security.useFIPS property must be set to `true` to enable suite B. The property must be entered in the `ssl.client.props` file in the global properties section, preferably after com.ibm.security.useFIPS.
com.ibm.websphere.security.suiteB	false	Specifies the level of Suite B security standard to enable. Valid values include 128 and 192. To enable the com.ibm.security.useFIPS property. it must be set to `true`. The property must be entered in the `ssl.client.props` file in the global properties section, preferably after com.ibm.security.useFIPS.

Certificate creation properties:

Use certificate creation properties to specify the default self-signed certificate values for the major attributes of a certificate. You can define the distinguished name (DN), expiration date, key size, and alias that are stored in the keystore.

Table 99. Certificate creation properties. This table describes the certificate creation properties.

Property	Default	Description
com.ibm.ssl.defaultCertReqAlias	default_alias	This property specifies the default alias to use to reference the self-signed certificate that is created in the keystore. If the alias already exists with that name, the default alias is appended with _#, where the number sign (#) is an integer that starts with 1 and increments until it finds a unique alias.
com.ibm.ssl.defaultCertReqSubjectDN	cn=${*hostname*}, o=IBM,c=US	This property uses the property distinguished name (DN) that you set for the certificate when it is created. The ${*hostname*} variable is expanded to the host name on which it resides. You can use correctly formed DNs as specified by the X.509 certificate.
com.ibm.ssl.defaultCertReqDays	365	This property specifies the validity period for the certificate and can be as small as 1 day and as large as the maximum number of days that a certificate can be set, which is approximately 15 years.
com.ibm.ssl.defaultCertReqKeySize	1024	This property is the default key size. The valid values depend upon the Java Virtual Machine (JVM) security policy files that are installed. By default, the product JVMs ship with the export policy file that limits the key size to 1024. To get a large key size such as 2048, you can download the restricted policy files from the website.

Certificate revocation checking:

To enable certificate revocation checking, you can set a combination of Online Certificate Status Protocol (OCSP) properties. These properties are not used unless you set the com.ibm.ssl.trustManager property to `IbmPKIX`. In addition, to successfully process revocation checking on the client, you must turn off the signer exchange prompt. To turn off the signer exchange prompt, change the com.ibm.ssl.enableSignerExchangePrompt property to `false`. For more information, see the related link to the "Enabling certificate revocation checking with the default IbmPKIX trust manager" topic.

SSL configuration properties:

Use the SSL configuration properties section to set multiple SSL configurations. For a new SSL configuration specification, set the com.ibm.ssl.alias property because the parser starts a new SSL configuration with this alias name. The SSL configuration is referenced by using the alias property from another file, such as `sas.client.props` or `soap.client.props`, through the default alias property. The properties that are specified in the following table enable you to create a javax.net.ssl.SSLContext, among other SSL objects.

Table 100. SSL configuration properties. This table lists the SSL configuration properties.

Property	Default	Description
com.ibm.ssl.alias	DefaultSSLSettings	This property is the name of this SSL configuration and must be the first property for an SSL configuration because it references the SSL configuration. If you change the name of this property after it is referenced elsewhere in the configuration, the runtime defaults to the com.ibm.ssl.defaultAlias property whenever the reference is not found. The error `trust file is null` or `key file is null` might display when you start an application using an SSL reference that is no longer valid.

Table 100. SSL configuration properties (continued). This table lists the SSL configuration properties.

Property	Default	Description
com.ibm.ssl.protocol	SSL_TLS	This property is the SSL handshake protocol that is used for this SSL configuration. This property attempts Transport Layer Security (TLS) first, but accepts any remote handshake protocol, including SSLv3 and TLSv1. Valid values for this property include SSL_TLS, SSL, SSLv2 (client side only), SSLv3, TLS, TLSv1, SSL_TLSv2, TLSv1.1, and TLSv1.2.
com.ibm.ssl.securityLevel	STRONG	This property specifies the cipher group that is used for the SSL handshake. The typical selection is STRONG, which specifies 128-bit or higher ciphers. The MEDIUM selection provides 40-bit ciphers. The WEAK selection provides ciphers that do not perform encryption, but do perform signing for data integrity. If you specify your own cipher list selection, uncomment the property com.ibm.ssl.enabledCipherSuites. **Note:** The use of javax.net.ssl system properties causes this value to always be HIGH.
com.ibm.ssl.trustManager	IbmX509	This property specifies the default trust manager that you must use to validate the certificate sent by the target server. This trust manager does not perform certificate revocation list (CRL) checking. You can choose to change this value to IbmPKIX for CRL checking using CRL distribution lists in the certificate, which is a standard way to perform CRL checking. When you want to perform custom CRL checking, you must implement a custom trust manager and specify the trust manager in the com.ibm.ssl.customTrustManagers property. The IbmPKIX option might affect performance because this option requires IBMCertPath for trust validation. Use IbmX509 unless CRL checking is necessary. If you are using the Online Certificate Status Protocol (OCSP) properties, set this property value to `IbmPKIX`.
com.ibm.ssl.keyManager	IbmX509	This property specifies the default key manager to use for choosing the client alias from the specified keystore. This key manager uses the com.ibm.ssl.keyStoreClientAlias property to specify the keystore alias. If this property is not specified, the choice is delegated to Java Secure Socket Extension (JSSE). JSSE typically chooses the first alias that it finds.
com.ibm.ssl.contextProvider	IBMJSSE2	This property is used to choose the JSSE provider for the SSL context creation. It is recommended that you default to IBMJSSE2 when you use a Java virtual machine (JVM). The client plug-in can use the SunJSSE provider when using a Sun JVM.
com.ibm.ssl.enableSignerExchangePrompt	true	This property determines whether to display the signer exchange prompt when a signer is not present in the client truststore. The prompt displays information about the remote certificate so that WebSphere Application Server can decide whether or not to trust the signer. It is very important to validate the certificate signature. This signature is the only reliable information that can guarantee that the certificate has not been modified from the original server certificate. For automated scenarios, disable this property to avoid SSL handshake exceptions. Run the `retrieveSigners` script, which sets up the SSL signer exchange, to download the signers from the server prior to running the client. If you are using the Online Certificate Status Protocol (OCSP) properties, set this property value to `false`.
com.ibm.ssl.keyStoreClientAlias	default	This property is used to reference an alias from the specified keystore when the target does not request client authentication. When WebSphere Application Server creates a self-signed certificate for the SSL configuration, this property determines the alias and overrides the global com.ibm.ssl.defaultCertReqAlias property.

Table 100. SSL configuration properties (continued). This table lists the SSL configuration properties.

Property	Default	Description
com.ibm.ssl.customTrustManagers	Commented out by default	This property enables you to specify one or more custom trust managers, which are separated by commas. These trust managers can be in the form of *algorithm*|*provider* or *classname*. For example, IbmX509|IBMJSSE2 is in the *algorithm*|*provider* format, and the com.acme.myCustomTrustManager interface is in the *classname* format. The class must implement the javax.net.ssl.X509TrustManager interface. Optionally, the class can implement the com.ibm.wsspi.ssl.TrustManagerExtendedInfo interface. These trust managers run in addition to the default trust manager that is specified by the com.ibm.ssl.trustManager interface. These trust managers do not replace the default trust manager.
com.ibm.ssl.customKeyManager	Commented out by default	This property enables you to have one, and only one, custom key manager. The key manager replaces the default key manager that is specified in the com.ibm.ssl.keyManager property. The form of the key manager is *algorithm*|*provider* or *classname*. See the format examples for the com.ibm.ssl.customTrustManagers property. The class must implement the javax.net.ssl.X509KeyManager interface. Optionally, the class can implement the com.ibm.wsspi.ssl.KeyManagerExtendedInfo interface. This key manager is responsible for alias selection.
com.ibm.ssl.dynamicSelectionInfo	Commented out by default	This property enables dynamic association with the SSL configuration. The syntax for a dynamic association is *outbound_protocol*, *target_host*, or *target_port*. For multiple specifications, use the vertical bar (|) as the delimiter. You can replace any of these values with an asterisk (*) to indicate a wildcard value. Valid *outbound_protocol* values include: IIOP, HTTP, LDAP, SIP, BUS_CLIENT, BUS_TO_WEBSPHERE_MQ, BUS_TO_BUS, and ADMIN_SOAP. When you want the dynamic selection criteria to choose the SSL configuration, uncomment the default property, and add the connection information. For example, add the following on one line `com.ibm.ssl.dynamicSelectionInfo=HTTP, .ibm.com,443|HTTP,.ibm.com,9443`
com.ibm.ssl.enabledCipherSuites	Commented out by default	This property enables you to specify a custom cipher suite list and override the group selection in the com.ibm.ssl.securityLevel property. The valid list of ciphers varies according to the provider and JVM policy files that are applied. For cipher suites, use a space as the delimiter.
com.ibm.ssl.keyStoreName	ClientDefaultKeyStore	This property references a keystore configuration name. If you have not already defined the keystore, the rest of the keystore properties must follow this property. After you define the keystore, you can specify this property to reference the previously specified keystore configuration. New keystore configurations in the `ssl.client.props` file have a unique name.
com.ibm.ssl.trustStoreName	ClientDefaultTrustStore	This property references a truststore configuration name. If you have not already defined the truststore, the rest of the truststore properties must follow this property. After you define the truststore, you can specify this property to reference the previously specified truststore configuration. New truststore configurations in the `ssl.client.props` file should have a unique name.

Keystore configurations:

SSL configurations reference keystore configurations whose purpose is to identify the location of certificates. Certificates represent the identity of clients that use the SSL configuration. You can specify keystore configurations with other SSL configuration properties. However, it is recommended that you

specify the keystore configurations in this section of the `ssl.client.props` file after the com.ibm.ssl.keyStoreName property identifies the start of a new keystore configuration. After you fully define the keystore configuration, the com.ibm.ssl.keyStoreName property can reference the keystore configuration at any other point in the file.

Table 101. Keystore configuration properties. This table lists the keystore configuration properties.

Property	Default	Description
com.ibm.ssl.keyStoreName	ClientDefaultKeyStore	This property specifies the name of the keystore as it is referenced by the runtime. Other SSL configurations can reference this name further down in the `ssl.client.props` file to avoid duplication.
com.ibm.ssl.keyStore	${user.root}/etc/key.p12	This property specifies the location of the keystore in the required format of the com.ibm.ssl.keyStoreType property. Typically, this property references a keystore file name. However, for cryptographic token types, this property references a Dynamic Link Library (DLL) file. **gotcha:** If you are using a 4764 cryptography card, then the keystore file name for the client configuration should be specified as the file `4764.cfg` in a directory structure of your choice, and the corresponding com.ibm.ssl.keyStoreType should be set to PKCS11. The 4764.cfg file is NOT supplied with WebSphere Application Server.
com.ibm.ssl.keyStorePassword	WebAS	This property is the default password, which is the cell name for the profile when it is created. The password is typically encoded using an {xor} algorithm. You can use iKeyman to change the password in the keystore, then change this reference. If you do not know the password and if the certificate is created for you, change the password in this property, then delete the keystore from the location where it resides. Restart the client to recreate the keystore by using the new password, but only if the keystore name ends with DefaultKeyStore and if the fileBased property is `true`. Delete both the keystore and truststore at the same time so that a proper signer exchange can occur when both are recreated together.
com.ibm.ssl.keyStoreType	PKCS12	This property is the keystore type. Use the default, PKCS12, because of its interoperability with other applications. You can specify this property as any valid keystore type that is supported by the JVM on the provider list.
com.ibm.ssl.keyStoreProvider	IBMJCE	The IBM Java Cryptography Extension property is the keystore provider for the keystore type. The provider is typically IBMJCE or IBMPKCS11Impl for cryptographic devices.
com.ibm.ssl.keyStoreFileBased	true	This property indicates to the runtime that the keystore is file-based, meaning it is located on the file system.
com.ibm.ssl.keyStoreReadOnly	false	This property indicates to the run time for WebSphere Application Server whether the key store can be modified during the run time.

Truststore Configurations:

SSL configurations reference truststore configurations, whose purpose is to contain the signer certificates for servers that are trusted by this client. You can specify these properties with other SSL configuration properties. However, it is recommended that you specify truststore configurations in this section of the

ssl.client.props file after the com.ibm.ssl.trustStoreName property has identified the start of a new truststore configuration. After you fully define the truststore configuration, the com.ibm.ssl.trustStoreName property can reference the configuration at any other point in the file.

A truststore is a keystore that JSSE uses for trust evaluation. A truststore contains the signers that WebSphere Application Server requires before it can trust the remote connection during the handshake. If you configure the com.ibm.ssl.trustStoreName=ClientDefaultKeyStore property, you can reference the keystore as the truststore. Further configuration is not required for the truststore because all of the signers that are generated through signer exchanges are imported into the keystore where they are called by the runtime.

Table 102. Truststore Configuration properties. This table lists the truststore configuration properties.

Property	Default	Description
com.ibm.ssl.trustStoreName	ClientDefaultTrustStore	This property specifies the name of the truststore as it is referenced by the runtime. Other SSL configurations can reference further down in the ssl.client.props file to avoid duplication.
com.ibm.ssl.trustStore	${user.root}/etc/trust.p12	This property specifies the location of the truststore in the format that is required by the truststore type that is referenced by the com.ibm.ssl.trustStoreType property. Typically, this property references a truststore file name. However, for cryptographic token types, this property references a DLL file. **gotcha:** If you are using a 4764 cryptography card, then the keystore file name for the client configuration should be specified as the file 4764.cfg in a directory structure of your choice, and the corresponding com.ibm.ssl.keyStoreType should be set to PKCS11. The 4764.cfg file is NOT supplied with WebSphere Application Server.

Table 102. Truststore Configuration properties (continued). This table lists the truststore configuration properties.

Property	Default	Description
com.ibm.ssl.trustStorePassword	WebAS	This property specifies the default password, which is the cell name for the profile when it is created. The password is typically encoded using an {xor} algorithm. You can use iKeyman to change the password in the keystore, then change the reference in this property. If you do not know the password and if the certificate was created for you, change the password in this property, then delete the truststore from the location where it resides. Restart the client to recreate the truststore by using the new password, but only if the keystore name ends with DefaultTrustStore and the fileBased property is `true`. It is recommended that you delete the keystore and the truststore at the same time so that a proper signer exchange can occur when both are recreated together.
com.ibm.ssl.trustStoreType	PKCS12	This property is the truststore type. Use the default PKCS12 type because of its interoperability with other applications. You can specify this property as any valid truststore type that is supported by the JVM functionality on the provider list.
com.ibm.ssl.trustStoreProvider	IBMJCE	This property is the truststore provider for the truststore type. The provider is typically IBMJCE or IBMPKCS11Impl for cryptographic devices.
com.ibm.ssl.trustStoreFileBased	true	This property indicates to the runtime that the truststore is file-based, meaning it is located on the file system.
com.ibm.ssl.trustStoreReadOnly	false	This property indicates to the run time for WebSphere Application Server whether the truststore can be modified during the run time.

Creating a CA client in SSL

A plug point is provided to allow users to connect to a certificate authority (CA) to request, query, and revoke certificates. A security configuration object, called a CAClient, must be created for WebSphere to communicate with the CA. The CAClient object must contain a WSPKIClient() implementation, and it will handle the connection and communicate with the CA server. Users can also create there own implementation.

Before you begin

The WSPKIClient interface must be implemented and the class name provided as part of the CAClient when it is created.

You use the administrative console to create a new CA client.

Procedure

1. Click **Security > SSL certificate and key management**.
2. Click **Certificate Authority (CA) client configurations**. A panel of existing CA clients appears.
3. Click **New** to create a new CA client in the configuration.

   Note: You can also create a CA client by using the **createCAClient** AdminTask .
4. Fill in the following information for the CA client
   - Name of the CA client.
   - The management scope (selected from the drop-down list).
   - WSPKIClient implementation class.
   - CA server host name.
   - User name.
   - Password.
   - Confirm of password.
   - Number of times to poll.
   - Polling interval (in minutes) when requestin certificates.
   - Custom properties.
5. Click **Apply** then **OK**.

Results

The information in the object can then be used by the runtime to connect to a CA to create, revoke, or replace a certificate.

Deleting a CA client in SSL

You can delete the CAClient object from the security configuration if a connection to a certificate authority (CA) is no longer needed.

Before you begin

You use the administrative console to delete a CA client.

Procedure

1. Click **Security > SSL certificate and key management**.
2. Click **Certificate Authority (CA) client configurations**. A panel displaying the existing CA clients appears.
3. Click the CA client name you want to delete.
4. Click the **Delete** button.

   Note: You can also use the **deleteAClient** AdminTask to delete the CA client.

Results

The CA client is deleted from the configuration.

Note: When you use the deleteCAClient AdminTask to delete the CA client, the CA client cannot be deleted if a CA certificate that exists in the keystore was obtained from the certificate authority and is still referenced by the CA client. For example, when such CA certificate still exists, the user receives the following message:

```
wsadmin>$AdminTask deleteCAClient {-caClientName myca}
WASX7015E: Exception running command:
 "$AdminTask deleteCAClient {-caClientName myca}"; exception information:
 com.ibm.websphere.management.cmdframework.CommandValidationException:
 CWPKI0687E: The Certificate Authority (CA) client myca is still referenced by:
 [Certificate alias myca21 in key store CellDefaultKeyStore].
wsadmin>
```

Viewing or modifying a CA client in SSL

You can view or modify the CAClient object settings in the security configuration. The CAClient object contains all the information needed to connect and communicate with a certificate authority (CA). A connection to a Certificate Authority is used to request a certificate, query a certificate, or revoke a certificate.

Before you begin

You use the administrative console to view or modify a CA client.

Procedure

1. Click **Security > SSL certificate and key management**.
2. Click **Certificate Authority (CA) client configurations**. A panel displaying the existing CA clients appears.
3. Click the CA client name you want to examine and modify.

   Note: You can also use the **getCAClient** AdminTask to get information about the existing CA client and the **modifyCACleint** AdminTask to make changes to the CA client.
4. Make the changes to the CA client information as required. Modify the following information as required.
   - Name of the CA client.
   - The management scope (selected from the drop-down list.
   - Implementation class.
   - CA server host name.
   - User name.
   - Password.
   - Confirm of password.
   - Number of times to poll.
   - Polling interval (in minutes) when requestin certificates.
   - Custom properties.
5. Click **Apply** then **OK**.

Results

The information in the object can then be used by the runtime to connect to a CA to create, revoke, or replace a certificate

What to do next

Creating a keystore configuration for a preexisting keystore file

A Secure Sockets Layer (SSL) configuration references keystore configurations during security processing. If another keystone tool is used to create a keystore file, or the keystone file was saved from a previous configuration, you must create a new keystone configuration object that references the preexisting keystone file. The server then uses this new keystone configuration object to obtain information from the preexisting keystone file.

Before you begin

A keystore must already exist.

Alternative Method: To create a keystore by using the wsadmin tool, use the **createKeyStore** command of the AdminTask object. For more information, see the KeyStoreCommands command group for the AdminTask object article.

About this task

Complete the following steps in the administrative console:

Procedure

1. Click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound}**.
2. Under Related Items, click **Key stores and certificates**, then click **New**.
3. Type a name in the **Name** field. This name uniquely identifies the keystore in the configuration.
4. Type the location of the keystore file in the **Path** field. The location can be a file name or a file URL to an existing keystore file.
5. Type the keystore password in the **Password** field. This password is for the keystore file that you specified in the **Path** field.
6. Type the keystore password again in the **Confirm Password** field to confirm the password.
7. Select a keystore type from the list. The type that you select is for the keystore file that you specified in the **Path** field.
8. Select any of the following optional selections:
   - The **Read only selection** creates a keystore configuration object but does not create a keystore file. If this option is selected, the keystore file that you specified in the **Path** field must already exist.
   - The **Initialize at startup selection** initializes the keystore during runtime.
   - The **Enable cryptographic operations on a hardware device** specifies whether a hardware cryptographic device is used for cryptographic operations only.

     gotcha: Operations that require login are not supported when using this option.
9. Click **Apply** and **Save**.

Results

You have created a keystore configuration object for the keystore file that you specified. This keystore can now be used in an SSL configuration.

Configuring a hardware cryptographic keystore

You can create a hardware cryptographic keystore that WebSphere Application Server can use to provide cryptographic token support in the server configuration.

About this task

Note: The hardware accelerator is not supported except for the following situations:

- If you are using WebSphere Application Server for z/OS and are using the IBMJCECCA crypto provider.
- If you are using WebSphere Application Server Version 7.0 and above running on zLinux and are using the IBMPKCS11 provider.

Complete the following steps in the administrative console:

Procedure

1. Click **Security > SSL certificate and key management > Key stores and certificates**.
2. Click **New**.
3. Type a name to identify the keystore. This name is used to enable hardware cryptography in the Web Services Security configuration.
4. Optionally, you can type a description for the keystore in the **Description** field.
5. You can specify a **Management scope** for the key store. This is not required. The management scope specifies the scope where this Secure Sockets Layer (SSL) configuration is visible. For example, if you choose a specific node, then the configuration is only visible on that node and any servers that are part of that node.
6. Type the path for the hardware device-specific configuration file. The configuration file is a text file that contains entries in the following format: *attribute* = *value*. The valid values for attribute and value are described in detail in the Software Developer Kit, Java Technology Edition documentation. The two mandatory attributes are name and library, as shown in the following sample code:

   ```
   name = FooAccelerator
   library = /opt/foo/lib/libpkcs11.so
   slotListIndex = 0
   ```

   The configuration file should also include device-specific configuration data. Navigate to the PKCS11ImplConfigSamples.jar file, which contains sample configuration files, under the heading "PKCS 11 Implementation Provider" on the Java technology site http://www.ibm.com/developerworks/java/jdk/security/60/.

   Note: JSSE2 is unable to use the IBMPKCS11Impl provider for acceleration.

   a. You can use this link http://www.ibm.com/developerworks/java/jdk/security/50/secguides/pkcs11implDocs/IBMJavaPKCS11ImplementationProvider.html to initialize the IBMPKCS11 provider in a thread safe way
   b. Specify a unique .cfg file that contains information about the supported hardware device. A list of supported hardware devices are available at http://www.ibm.com/developerworks/java/jdk/security/50/secguides/pkcs11implDocs/IBMPKCS11SupportList.html
   c. You specify the Signature.getInstance method with the properly initialized IBMPKCS11Impl provider instance as shown.

```
Signature.getInstance("SHA1withRSA", ibmpkcs11implinstance);
```

7. Type a password if the token login is required. Operations that use keys on the token require a secure login. This field is optional if the keystore is used as a cryptographic accelerator. In this case, you need to select **Enable cryptographic operations on hardware device**.
8. Select the **PKCS11** type.
9. Select **Read only**.
10. Click **OK** and **Save**.

Results

WebSphere Application Server can now provide cryptographic token support in the server configuration.

Managing keystore configurations remotely

You can manage keystores remotely in a WebSphere Application Server, Network Deployment environment on separate machines. A node server can hold the configuration for a keystore, while the actual keystore resides on another system. After you set up a remotely managed configuration, you can perform all of the certificate and keystore operations for the keystore on the remote machine from the server that contains the keystore remote configuration.

Before you begin

Key stores can be remotely managed only in network deployed environments.

Alternative Method: To manage a self-signed certificates by using the wsadmin tool, use the **PersonalCertificateCommands** group commands of the AdminTask object. For more information, see the PersonalCertificateCommands command group for the AdminTask object article.

About this task

Complete the following steps in the administrative console:

Procedure

1. Click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > *ssl_configuration* > Key stores and certificates**.
2. Click **New**.
3. Type a name in the **Name** field. This name uniquely identifies the keystore in the configuration.
4. Type the location of the keystore file in the **Path** field. The location can be a file name or a file Uniform Resource Locator (URL) to an existing keystore file.
5. Type the keystore password in the **Password** field. This password is for the keystore file that you specified in the **Path** field.
6. Type the keystore password again in the **Confirm Password** field to confirm the password.
7. Select a keystore type from the list. The type you select is for the keystore file that you specified in the **Path** field.
8. Select the **Remotely managed** check box, and then fill in one or more hosts names of the systems where the keystore file is to be located. If you provide multiple host names, separate the host names with a pipe (**|**).
9. Select any of the following optional selections:
   - The **Read only selection** creates a keystore configuration object but does not create a keystore file. If this option is selected, the keystore file that you specified in the **Path** field must already exist.
   - The **Initialize at startup selection** initializes the keystore during run time.
10. Select **Apply** and **Save**.

Results

A keystore configuration object is created on the server from where the command was run. The keystore file for the configuration will be created on each system that you specified in the host list.

What to do next

Now, you can perform all certificate management operations on the keystore from the system where the keystore configuration resides. For example, you can perform certificate management operations, such as: creating a self-signed certificate, extracting a certificate, or extracting a signer certificate.

Keystores and certificates collection

Use this page to manage keystore types, including cryptography, Resource Access Control Facility (RACF), Certificate Management Services (CMS), Java, and all trust store types.

gotcha: In most cases, having unused and expired signer certificates in a trust store does not cause problems. However, if you experience a problem because the trust store includes an unused or expired signer certificate, you can safely delete the following expired signer certificates from the dummy keystores files:

- DummyClientKeyFile.jks
- DummyClientTrustFile.jks
- DummyServerKeyFile.jks
- DummyServerTrustFile.jks

To view this administrative console page, click **Security > SSL certificate and key management**. Under Configuration settings, click **Manage endpoint security configurations > {Inbound | Outbound} > *ssl_configuration***. Under Related items, click **Keystores and certificates**.

Table 103. Keystores and certificates buttons. This table describes the keystores and certificates buttons.

Button	Resulting action
New	Adds a new keystore object that can be referenced by Secure Sockets Layer (SSL) configurations or KeySets. The Keystore management scope is based on the part of the topology tree from which it was created.
Delete	Deletes an existing keystore. The keystore should not be referenced by any other parts of the configuration before you delete it.
Change password	Allows for changing a keystore password.
Exchange signers	Refers to exchanging signers in a keystore. You can select two keystores, along with personal certificates or signer certificates from a selected keystore, then add them as a signer to another selected keystore.

Keystore usages

Filters the keystore usage types in the keystore collection.

The default value for the keystore usage filter depends on the navigation path that you followed to get to the Keystores and certificates panel. You can change the value of the keystore usage filter by clicking on the drop-down list and selecting a different filter value.

Navigation path	Keystore usage default value
Security > SSL certificate and key management > Keystores and certificates	SSL keystores
Security > SSL certificate and key management > Key sets > CellLTPAKeyPair > Keystores and certificates	Key set keystores
Security > SSL certificate and key management > SSL configurations > CellDefaultSSLSettings > Keystores and certificates	SSL keystores
Security > SSL certificate and key management > Manage endpoint security configurations > *node name* > Keystores and certificates	SSL keystores

Name
Specifies the unique name that is used to identify the keystore. This name is typically scoped by the ManagementScope scopeName and based upon the location of the keystore. The name must be unique within the existing keystore collection.

This is a user-defined name.

Description
Specifies the description of the keystore.

This is a user-defined description.

Path
Specifies the location of the keystore file in the format needed by the keystore type. This file can be a card-specific configuration file for cryptographic devices or a filename or file URL for file-based keystores. It can be a safkeyring URL for RACF keyrings.

Key store settings
Use this page to create all keystore types, including cryptographic, Resource Access Control Facility (RACF), Certificate Management Services (CMS), Java, and all truststore types.

To view this administrative console page, click **Security > SSL certificate and key management**. Under Configuration settings, click **Manage endpoint security configurations > {Inbound | Outbound}**. Under Related Items, click **Key stores and certificates**. Click either **New** or an existing keystore.

Links to Personal certificates, Signer certificates, and Personal certificate requests enable you to manage certificates in a manner similar to iKeyman capabilities. A keystore can be file-based, such as CMS or Java keystore types, or it can be remotely managed.

Note: Any changes made to this panel are permanent.

Name
Specifies the unique name to identify the keystore. The keystore is typically scoped by the ManagementScope scopeName based on the location of the keystore. The name must be unique within the existing keystore collection.

Information	Value
Data type:	Text

Description
Specifies the description of the keystore.

Information	Value
Data type:	Text

Management scope
Specifies the scope where this Secure Sockets Layer (SSL) configuration is visible. For example, if you choose a specific node, then the configuration is only visible on that node and any servers that are part of that node.

Information	Value
Data type:	Text

Path
Specifies the location of the keystore file in the format needed by the keystore type. This file can be a dynamic link library (DLL) for cryptographic devices or a filename or file URL for file-based keystores. It can be a safkeyring URL for RACF keyrings.

Information	Value
Data type:	Text

Control region user
Specifies the Control region Started Task user ID in which the Control region System Authorization Facility (SAF) keyring is created. The user ID must match the exact ID being used by the Control region. Note: This option only applies when creating writable SAF keyrings on z/OS.

Information	Value
Data type:	Text

Servant region user
Specifies the Servant region Started Task user ID in which the Servant region System Authorization Facility (SAF) keyring is created. The user ID must match the exact ID being used by the Servant region. Note: This option only applies when creating writable SAF keyrings on z/OS.

Information	Value
Data type:	Text

Password [new keystore] | Password [existing keystore]
Specifies the password used to protect the physical keystore in the operating system. For the default keystore (names ending in DefaultKeyStore or DefaultTrustStore), the password is `WebAS`. This default password must be changed.

This field can be edited.

Information	Value
Data type:	Text

Note: If you want to push the key store to all nodes, the path should be: `${CONFIG_ROOT}/cells/CELLNAME/yourkeystore.kdb`.

Confirm password
Specifies confirmation of the password to open the keystore file or device.

Information	Value
Data type:	Text

Type
Specifies the implementation for keystore management. This value defines the tool that operates on this keystore type.

The list of options is returned by java.security.Security.getAlgorithms("KeyStore"). Some options might be filtered and some might be added based on the java.security configuration.

Information	Value
Data type:	Text
Default:	PKCS12

Read only
Specifies whether the keystore can be written to or not. If the keystore cannot be written to, certain operations cannot be performed, such as creating or importing certificates.

Information	Value
Default:	Disabled

Remotely managed
Specifies whether the key store is remotely managed, which means that a remote MBean call is needed to update the key store based on the host name specified in the host list field. Most hardware cryptographic token devices are remotely managed. If a key store is marked remotely managed, list the host name of the server where the device is installed in the Host list field.

Information	Value
Default:	

Initialize at startup
Specifies whether the keystore needs to be initialized before it can be used for cryptographic operations. If enabled, the keystore is initialized at server startup.

Information	Value
Default:	Disabled

Enable cryptographic operations on hardware device
Specifies whether a hardware cryptographic device is used for cryptographic operations only. Operations that require a login are not supported when using this option.

Information	Value
Default:	Disabled

Key managers collection
Use this page to define the implementation settings for key managers. A key manager is invoked during a Secure Sockets Layer (SSL) handshake to determine which certificate alias is used. The default key manager (WSX509KeyManager) performs alias selection. If more advanced function is desired, define a custom key manager on the **Manage endpoint security configurations** panel.

To view this administrative console page, click **Security > SSL certificate and key management**. Under Configuration settings, click **Manage endpoint security configurations > {Inbound | Outbound} > *ssl_configuration***. Under Related items, click **Key managers**.

Table 104. Key managers buttons. This table describes the key managers buttons.

Button	Resulting action
New	Adds a new key manager that can be selected by an SSL configuration. A key manager is invoked during an SSL handshake to select a specific certificate alias to use from a key store.
Delete	Deletes an existing key manager. The key manager should not be referenced by any SSL configuration before you can delete it.

Name
Specifies the name of the key manager, which you can select on the SSL configuration panel.

Class name
Specifies the name of the key manager implementation class. This class implements javax.net.ssl.X509KeyManager interface and, optionally, the com.ibm.wsspi.ssl.KeyManagerExtendedInfo interface.

Algorithm
Specifies the algorithm name of the key manager that is implemented by the selected provider.

Key managers settings
Use this page to define key managers implementation settings. A key manager gets invoked during an Secure Sockets Layer (SSL) handshake to determine the certificate alias to be used. The default key manager (WSX509KeyManager) performs alias selection. If more advanced function is desired, a custom key manager can be specified here and selected in the SSL configuration.

To view this administrative console page, click **Security** > **SSL certificate and key management**. Under Configuration settings, click**Manage endpoint security configurations** > **{Inbound | Outbound}** > ***ssl_configuration***. Under Related items, click **Key managers**. On the next panel, click **New**.

Name
Specifies the name of the key manager, which you can select on the SSL configuration panel.

Information	Value
Data type:	Text

Management scope
Specifies the scope where this Secure Sockets Layer (SSL) configuration is visible. For example, if you choose a specific node, then the configuration is only visible on that node and any servers that are part of that node.

Information	Value
Data type	List
Range:	Applicable scopes

Standard
Specifies the key manager selection that is available from a Java provider that is installed in the java.security file. This provider might be shipped by Java Secure Sockets Extension (JSSE) or be a custom provider that implements an X509KeyManager interface.

Information	Value
Default:	Enabled

Provider
Specifies the provider name that has an implementation of an X509KeyManager interface. This provider is typically set to IBMJSSE2.

Information	Value
Data type:	Text
Default:	IBMJCE

Algorithm
Specifies the algorithm name of the trust manager implemented by the selected provider.

Information	Value
Data type:	Text
Default:	IbmX509

Custom
Specifies that the key manager selection is based on a custom implementation class that implements the javax.net.ssl.X509KeyManager interface and optionally the com.ibm.wsspi.ssl.KeyManagerExtendedInfo interface to obtain additional connection information not otherwise available.

Information	Value
Default:	Disabled

Class name
Specifies the name of the key manager implementation class.

Information	Value
Data type:	Text

Creating a self-signed certificate

You can create a self-signed certificate. WebSphere Application Server uses the certificate at runtime during the handshake protocol. Self-signed certificates are located in the default keystore.

Before you begin

You must create a keystore before you can create a self-signed certificate.

Alternative Method: To create a self-signed certificate by using the wsadmin tool, use the **createSelfSignedCertificate** command of the AdminTask object. For more information, see the PersonalCertificateCommands command group for the AdminTask object article.

About this task

Complete the following steps in the administrative console:

Procedure

1. Click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > *ssl_configuration* > Key stores and certificates > [*keystore*]**.
2. From Additional Properties, click **Personal certificates**.
3. Click **Create a self-signed certificate**.
4. Type a certificate alias name. The alias identifies the certificate request in the keystore.
5. Type a common name (CN) value. This value is the CN value in the certificate distinguished name (DN).
6. Type the validity period The default validity period value is 365 days.
7. You can configure one or more of the following optional values:
   a. Optional: Select a key size value. The default key size value is 2048 bits.
   b. Optional: Type an organization value. This value is the O value in the certificate DN.

c. Optional: Type an organizational unit value. This organizational unit value is the OU value in the certificate DN.
d. Optional: Type a locality value. This locality value is the L value in the certificate DN.
e. Optional: Type a state or providence value. This value is the ST value in the certificate DN.
f. Optional: Type a zip code value. This zip code value is the POSTALCODE value in the certificate DN.
g. Optional: Select a country value from the list. This country value is the C= value in the certificate request DN.

8. Click **Apply**.

Results

You have created a self-signed certificate that resides in the keystore. The SSL configuration for the WebSphere Application Server runtime uses this certificate for SSL communication. Extract the signer of the self-signed certificate to add the signer to another keystore.

Replacing an existing personal certificate

Occasionally, you need to replace an existing personal certificate with a new certificate. This task discusses how to replace the existing personal certificate in the keystore. It searches all keystores for a signer certificate extracted from the original personal certificate, and places the signer of the new personal certificate in it's place. It also updates all of the certificate alias references in the security configuration with the new one.

Before you begin

The current certificate and the certificate replacement must exist in the same keystore before you can replace a certificate.

Alternative Method: To replace a self-signed certificate by using the wsadmin tool, use the **replaceCertificate** command of the AdminTask object. For more information, see the PersonalCertificateCommands command group for the AdminTask object article

About this task

Complete the following steps in the administrative console:

Procedure

1. Click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > *ssl_configuration* > Key stores and certificates > [*keystore*]**.
2. Under Additional Properties, click **Personal certificates**.
3. Select the certificate to be replaced. The alias list must include the certificate to be replaced and the certificate to replace it with.
4. Click **Replace**.
5. Select a replacement certificate alias from the list.
6. You can delete one of the following types of certificates:
   - Select **Delete old certificate** to delete the existing or expired certificate.
   - Select **Delete old signers** to delete the existing signer certificates.
7. Click **Apply**.

Results

Your results depend on what you selected:

- If you selected **Delete old certificate**, the new certificate alias replaces all of the references to the certificate alias in the configuration.
- If you selected **Delete old signers**, the new signer certificate replaces all of the occurrences of the old signer certificates.
- If the new certificate alias replaces the existing alias, the WebSphere Application Server runtime checks to make sure that:
  - All of the SSL Configurations objects reference the certificate
  - The Dynamic SSL Configuration Selections objects and the SSL Configuration group objects reference the certificate.
- If you selected **Delete old signers**, the existing signer certificates are replaced.
- If you selected **Delete old certificate**, the existing certificate is deleted.

Creating a new SSL certificate to replace an existing one in a node

When using the -asExistingNode option on the addNode command, you might be adding an existing node to a different machine. The default Secure Sockets Layer (SSL) certificate of the node does not contain the name of the machine the node is located on. In most scenarios, the subject DN of the default certificate does not make a difference. However, you might want to change the default certificate of the node to contain the hostname of the node.

Before you begin

To replace the default certificate of a node, you must create a new NodeDefaultKeyStore for the certificate and then replace the old certificate with the new one.

The certificate created by default on the WebSphere Application Server subjectDN is of the form cn=<hostname>, ou=<cell name>, ou=<node name>, o=ibm, c=us. When creating a new certificate you can also customize the subjectDN.

About this task

To create a new SSL certificate in the administrative console:

Procedure

1. Click **Security > SSL certificate and key management > Key stores and certificates**.
2. Select the NodeDefaultKeyStore of the node you want to change.
3. Under Additional Properties, select **Personal certificates**.
4. Under the Create pull-down, select **Chained Certificate**.
5. Enter a certificate and alias name. This can be any name you choose as long as the alias does not already exist. It is just a label to identify the certificate in the keystore.
6. Enter a common name. This is typically the hostname the node is running on.
7. Optional: Fill in any of the other Subject DN related fields. If you want the subject DN to look like the default subjectDN on WebSphere Application Server, then enter:
   - `IBM` in the Organization field.
   - `<cell name>,ou=<node name>` in the Organization unit field.
   - Under the Country or region pull-down, select **US**.
8. You can use the defaults for Root certificate used to sign the certificate, Key Size, and Validity Period or supply your own values.
9. Click **Apply**.

Note: You can also create a new chained certificate using the createChainedCertificate command. Read PersonalCertificateCommands command group for the AdminTask object for more information.

You must now replace the old certificate with the one you just created. The replace certificate option not only replaces the old default certificate with a new one but also replaces any occurrences of the signer of the old certificate with the signer of the new certificate. The configuration is also checked for references to the alias name of the old certificate and replaces it with the alias name of the new certificate. To replace the old certificate with the new one, complete the remaining steps.

10. Click **Security > SSL certificate and key management > Key stores and certificates**.
11. Select the NodeDefaultKeyStore of the node you want to change.
12. Under Additional Properties, select **Personal certificates**.
13. Select the default certificate of the node, usually called `default`.
14. Click **Replace**.
15. Select the certificate alias name for the certificate you just created from the **Replace with** pull-down.
16. Click **Delete old Certificate after replacement**.
17. Click **Apply**.

Note: You can also create a new chained certificate using the replaceCertificate command. Read PersonalCertificateCommands command group for the AdminTask object for more information.

What to do next

You can also replace default certificates in an entire cell. Read Creating new SSL certificates to replace existing ones in a cell for more information.

Creating new SSL certificates to replace existing ones in a cell

To replace default Secure Socket Layer (SSL) certificates in an entire cell, you must create a new self-signed root certificate in the root keystore, DmgrDefaultRootStore, and replace the old root certificate with the new one.

About this task

For the default certificate of the cell in CellDefaultKeyStore and the default certificate of each node in NodeDefaultKeyStore, create a new chained certificate and replace the old default certificate with the new certificate.

The root certificate is created by default on WebSphere Application Server, and has a subjectDN in the form cn=<hostname>, ou=Root Certificate, ou=<cell name>, ou=<node name>, o=ibm, c=us. When you create a new root certificate you can also customize the subject DN.

To create a new SSL root certificate in the administrative console:

Procedure

1. Click **Security > SSL certificate and key management > Key stores and certificates**.
2. Under the Keystore usages pull-down, select **Root certificate keystore**.
3. Select the DmgrDefaultRootStore in the keystore collection.
4. Under Additional Properties, select **Personal certificates**.
5. Under the Create pull-down, select **Self-signed Certificate**.
6. Enter a certificate and alias name. This can be any name you choose as long as the alias does not already exist. It is just a label to identify the certificate in the keystore.
7. Enter a common name. This is typically the hostname the node is running on.

8. Optional: Fill in any of the other Subject DN related fields. If you want the subject DN to look like the default subjectDN on WebSphere Application Server, then enter:
   - `IBM` in the Organization field.
   - `<cell name>,ou=<node name>` in the Organization unit field.
   - Under the Country or region pull-down, select **US**.
9. You can use the defaults for Root certificate used to sign the certificate, Key Size, and Validity Period or supply your own values.
10. Click **Apply**.

    Note: You can also create a self-signed certificate using the createSelfSignedCertificate command. Read PersonalCertificateCommands command group for the AdminTask object for more information.

    You must now replace the old root certificate with the one you just created. The replace certificate option not only replaces the old default certificate with a new one but also replaces any occurrences of the signer of the old certificate with the signer of the new certificate. The configuration is also checked for references to the alias name of the old certificate and replaces it with the alias name of the new certificate. To replace the old certificate with the new one, complete the remaining steps.
11. Click **Security > SSL certificate and key management > Key stores and certificates**.
12. Select the CellDefaultKeyStore of the node you want to change.
13. Under Additional Properties, select **Personal certificates**.
14. Select the default certificate of the node, usually called `default`.
15. Click **Replace**.
16. Select the certificate alias name for the new certificate you just created from the **Replace with** pull-down.
17. Click **Delete old Certificate after replacement**.
18. Click **Apply**.

What to do next

You can also replace default certificates in a node. Read Creating a new SSL certificate to replace an existing one in a node for more information

Creating a certificate authority request

To ensure Secure Sockets Layer (SSL) communication, servers require a personal certificate that is either self-signed, chained or signed by an external certificate authority (CA). You must first create a personal certificate request to obtain a certificate that is signed by a CA.

Before you begin

The keystore that contains a personal certificate request must already exist.

Alternative Method: To create a certificate request by using the wsadmin tool, use the **createCertificateRequest** command of the AdminTask object. For more information, see the CertificateRequestCommands command group of the AdminTask object article.

About this task

Complete the following steps in the administrative console:

Procedure

1. Click **Security > SSL certificate and key management > Key stores and certificates > *keystore***.
2. Click **Personal certificate requests > New**.
3. Type the full path of the certificate request file. The certificate request is created in this location.
4. Type an alias name in the **Key label** field. The alias identifies the certificate request in the keystore.
5. Type a common name (CN) value. This value is the CN value in the certificate distinguished name (DN).
6. You can configure one or more of the following optional values:
   a. Optional: Select a key size value. The valid key size values are 512, 1024, 2048, 4096, and 8192. The default key size value is 2048 bits.
   b. Optional: Type an organization value. This value is the O value in the certificate DN.
   c. Optional: Type an organizational unit value. This organizational unit value is the OU value in the certificate DN.
   d. Optional: Type a locality value. This locality value is the L value in the certificate DN.
   e. Optional: Type a state or providence value. This value is the ST value in the certificate DN.
   f. Optional: Type a zip code value. The zip code value is the POSTALCODE value in the certificate DN.
   g. Optional: Select a country value from the list. This country value is the C= value in the certificate request DN.
7. Click **Apply**.

Results

The certificate request is created in the specified file location in the keystore. The request functions as a temporary placeholder for the signed certificate until you manually receive the certificate in the keystore.

Note: Key store tools (such as iKeyman and keyTool) cannot receive signed certificates that are generated by certificate requests from WebSphere Application Server. Similarly, WebSphere Application Server cannot accept certificates that are generated by certificate requests from other keystore utilities.

What to do next

Now you can receive the CA-signed certificate into the keystore to complete the process of generating a signed certificate for your server.

Certificate request settings

Use this page to verify the properties of a personal certificate request.

To view this administrative console page, click **Security > SSL certificate and key management**. Under Configuration settings, click **Manage endpoint security configurations > {Inbound | Outbound} > *ssl_configuration***. Under Related items, click **Key stores and certificates > *key store***. Under Additional Properties, click **Personal certificate requests > *certificate request*** .

File for certificate request

Specifies the fully qualified name of the file that contains the certificate request that can be sent to a certificate authority (CA) server.

Key label

Specifies the certificate alias name for the signer in the key store.

Key size
Specifies the size of the keys that are generated.

Requested by
Specifies the Subject distinguished name (DN) that represents the identity of the certificate request.

Fingerprint (SHA Digest)
Specifies the SHA hash of the personal certificate, which can be used to verify that the certificate has not been altered when it is used in a remote connection.

Signature algorithm
Specifies the algorithm used to sign the certificate.

Personal certificates collection
Use this page to manage personal certificates.

To view this administrative console page, click **Security > SSL certificate and key management**. Under Configuration settings, click **Manage endpoint security configurations > {Inbound | Outbound} > *ssl_configuration***. Under Related items, click **Key stores and certificates > *key store*** . Under Additional Properties, click **Personal certificates**.

The **Personal certificates** page lists all personal certificates in the selected key store. You can do most certificate management operations in this panel, including creating a new self-signed certificate, deleting a certificate, receiving one generated from a CA, replacing a certificate (simultaneous delete and create, replacing references across all key stores), extracting the signer, and importing or exporting a personal certificate.

Personal certificate requests are temporary place holders for certificates that will be signed by a certificate authority (CA).

The Key store collection must contain at least two key store files. You must select one file in order to replace, extract, or export a key store,

Table 105. Personal certificates buttons. This table lists the personal certificates buttons.

Button	Resulting action
Create (drop-down list)	Enables the application server to create the following certificates: • Self-signed Certificate • CA-signed Certificate • Chained Certificate
Delete	Specifies to delete a certificate from the key store. Be careful that the certificate alias is not referenced elsewhere in the Secure Sockets Layer configuration.
Receive a certificate from a certificate authority	Enables the application server to receive a certificate authority (CA)-generated certificate from a file to complete a certificate request.
Replace	Replaces a personal certificate with another personal certificate. All key stores in the configuration looking for signer certificate form the original personal certificate and replaces them with the new personal certificates signer. Any place in the security configuration where the certificate alias is referenced will be replaced with the new certificate alias.
Extract	Extracts the signer part of personal certificate from the key store and stores it to a file. The file can then be used to add the signer to another key store.
Import	Imports a certificate, including the private key, from a key store file or managed key store.
Export	Exports a certificate, including the private key, to a specified key store file or manage key store.
Revoke	Revokes a CA-signed certificate.
Renew	Renews a self signed or chained certificate.

Alias
Specifies the alias by which the personal certificate is referenced in the key store.

When you select an alias, the View Certificate panel opens.

Issued by
Specifies the distinguished name of the entity by which the certificate was issued. This name is the same as the issued-to distinguished name when the personal certificate is self-signed.

Issued to
Specifies the distinguished name of the entity to which the certificate was issued.

Serial number
Specifies the certificate serial number that is generated by the issuer of the certificate.

Expiration
Specifies the expiration date of the signer certificate for validation purposes.

Self-signed certificates settings
Use this page to create self-signed certificates.

To view this administrative console page, click **Security SSL certificate and key management**. Under Configuration settings, click **Manage endpoint security configurations**

{Inbound | Outbound} > ***ssl_configuration***. Under Related items, click **Key stores and certificates** *keystore*. Under Additional Properties, click **Personal certificates** > **Create (drop-down list)** > **Self-signed certificate**.

This same help file is available when you create a new certificate or view an existing certificate. The fields in this help file are described according to how they appear and are used on the administrative console.

Alias
Specifies the alias for the personal certificate in the keystore.

You enter the alias name for the personal certificate in the keystore when you are creating a certificate. *The alias name is read-only when you view an existing certificate.*

Information	Value
Data type:	Text

Version
Specifies the version of the personal certificate. Valid versions include X509 V3, X509 V2, or X509 V1. It is recommended to use X509 V3 certificates.

This field is read-only when you create or view a certificate.

Information	Value
Data type:	Text
Default:	X509 V3
Range:	

Key size
Specifies the key size of the private key that is used by the personal certificate.

When you are creating a certificate you can select the key size from the drop-down list. This field is read-only when you view a certificate.

Information	Value
Data type:	Integer
Default:	1024
Other valid key sizes:	512, 2048, 4096

Common name

Specifies the common name portion of the distinguished name (DN). It is recommended that this name be the host name of the machine on which the certificate resides. In some cases, the common name is used to login during Secure Socket Layer (SSL) certificate authentication; therefore, in some cases, this name might be used as a user ID for a local operating system registry.

When you create a new certificate you can enter the common name in this field. This field does not display when you view an existing certificate.

Information	Value
Data type:	Text

Serial number

Identifies the certificate serial number that is generated by the issuer of the certificate. When creating a certificate this field does not appear.

This field is read-only when you view an existing certificate.

Validity period

Specifies the length in days during which the certificate is valid. The default is 365 days. You can enter any number of days you wish.

This field is read-only when you view an existing certificate. This field displays a validity period as a range of days between two dates. For example, `Valid from March 16, 2008 to March 16, 2009`.

Information	Value
Data type:	Text

Organization

You enter the organization portion of the distinguished name. This field is optional.

This field displays only when you create a new certificate.

Information	Value
Data type:	Text

Organization unit

Specifies the organization unit portion of the distinguished name. This field is optional.

This field displays only when you create a new certificate.

Information	Value
Data type:	Text

Locality
Specifies the locality portion of the distinguished name. This field is optional.

This field displays only when you create a new certificate.

Information	Value
Data type:	Text

State/Province
Specifies the state portion of the distinguished name. This field is optional.

This field displays only when you create a new certificate.

Information	Value
Data type:	Text

Zip code
Specifies the zip code portion of the distinguished name. This field is optional.

This field displays only when you create a new certificate.

Information	Value
Data type:	Integer

Country or region
Select the country portion of the distinguished name from the drop-down list. This field is optional.

This field displays only when you create a new certificate.

Information	Value
Data type:	Text
Default:	(none)

Refer to http://www.iso.org/iso/en/prods-services/iso3166ma/02iso-3166-code-lists/list-en1.html for a list of ISO 3166 country codes.

Validity period
Identifies the length, in days, when the certificate is valid. The default is 365 days.

This field is read-only when you view an existing certificate and shows the start and end dates.

Issued to
Identifies the distinguished name of the entity to which the certificate was issued.

This field is read-only when you view an existing certificate.

Issued by
Identifies the distinguished name of the entity that issued the certificate. When the personal certificate is self-signed, this name is identical to the **Issued to** distinguished name.

This field is read-only when you view an existing certificate.

Fingerprint (SHA Digest)
Identifies the Secure Hash Algorithm (SHA hash) of the certificate, which can be used to verify the certificate's hash at another location, such as the client side of a connection.

This field is read-only when you view an existing certificate.

Signature algorithm
Identifies the algorithm used to sign the certificate.

This field is read-only when you view an existing certificate.

Personal certificate requests collection
Use this page to manage personal certificate requests. Personal certificate requests are temporary place holders for certificates that will be signed by a certificate authority (CA).

To view this administrative console page, click **Security > SSL certificate and key management**. Under Configuration settings, click **Manage endpoint security configurations > {Inbound | Outbound} > *ssl_configuration***. Under Related items, click **Key stores and certificates > *key store*** . Under Additional Properties, click **Personal certificate requests**.

A private key is generated during the certificate request generation, but only the certificate is sent to the CA. The CA generates a new certificate, signed by the CA. This can be added in the Personal Certificates panel.

Table 106. Personal certificate requests buttons. This table lists the personal certificate requests buttons.

Button	Resulting action
New	Creates a personal certificate request that can be given to a certificate authority to complete.
Delete	Deletes a personal certificate request.
Extract	Extracts a personal certificate request. Only one certificate request can be selected at a time.
Query	Queries a personal certificate request. Only one certificate request can be selected at a time.

gotcha:

- Any changes made to this panel are permanent.
- A Personal certificate request places a valid self-signed certificate in the keystore. This placeholder certificate is later replaced with the certificate that the Certificate Authority signs and returns. You must have a default certificate assigned on the SSL configuration. If a default certificate is not assign, when multiple personal certificates exist in a keystore and no default certificate is selected, the selection of a certificate within the SSL configuration keystore is random, which might cause SSL handshake errors.

Key label
Specifies the alias that represents the personal certificate request in the key store.

Requested by
Specifies the Subject distinguished name (DN) that represents the identity of the certificate request.

Personal certificate requests settings
Use this page to create a new certificate request that can be extracted and sent to a certificate authority (CA).

To view this administrative console page, click **Security** > **SSL certificate and key management**. Under Configuration settings, click **Manage endpoint security configurations** > **{Inbound | Outbound}** > ***ssl_configuration***. Under Related items, click **Key stores and certificates** > **key store**. Under Additional Properties, click `Personal certificates requests`. Then click the `New` button.

Personal certificate requests are temporary place holders for certificates that will be signed by a certificate authority (CA). The private key is generated during the certificate request generation, but only the certificate is sent to the CA. The CA generates a new certificate, signed by the CA.

Note: Any changes made to this panel are permanent.

File for certificate request

Specifies the fully qualified file name from which the certificate request is exported. This portion of the certificate request can be given to the certificate authority to generate the real certificate. After the real certificate is generated, you can perform a “Receive a certificate from a certificate authority” from the personal certificate collection view.

Information	Value
Data type:	String

Key label

Specifies the alias that represents the personal certificate request in the key store.

Information	Value
Data type:	String

Key size

Specifies the size of the keys that are generated.

Information	Value
Data type:	Integer
Default:	2048

Common name

Specifies the name of the entity that the certificate represents. This common name can represent a person, company, or machine. For web sites, the common name is frequently the DNS host name where the server resides.

Information	Value
Data type:	String

Organization

Specifies the organization portion of the distinguished name.

Information	Value
Data type:	String

Organizational unit

Specifies the organization unit portion of the distinguished name. This field is optional.

Information	Value
Data type:	String

Locality

Specifies the locality portion of the distinguished name. This field is optional.

Information	Value
Data type:	String

State or province
Specifies the state portion of the distinguished name. This field is optional.

Information	Value
Data type:	String

Zip code
Specifies the zip code portion of the distinguished name. This field is optional.

Information	Value
Data type:	Integer

Country or region
Specifies the country portion of the distinguished name.

Information	Value
Data type:	String

Refer to http://www.iso.org/iso/en/prods-services/iso3166ma/02iso-3166-code-lists/list-en1.html for a list of ISO 3166 country codes.

Extract certificate request
Use this page to extract a certificate request to a file so it can be sent to a certificate authority (CA).

To view this administrative console page, click **Security > SSL certificate and key management**. Under Configuration settings, click **Manage endpoint security configurations > {Inbound | Outbound} > *ssl_configuration***. Under Related items, click **Key stores and certificates > *key store*** . Under Additional Properties, click **Personal certificate requests > Extract**.

Key label
Specifies the alias that represents the personal certificate request in the key store.

File for certificate request
Specifies the filename where the extracted certificate request is placed.

Information	Value
Data type:	Text

Receiving a certificate issued by a certificate authority
When a certificate authority (CA) receives a certificate request, it issues a new certificate that functions as a temporary placeholder for a CA-issued certificate. A keystore receives the certificate from the CA and generates a CA-signed personal certificate that WebSphere Application Server can use for Secure Sockets Layer (SSL) security.

Before you begin

The keystore must contain the certificate request that was created and sent to the CA. Also, the keystore must be able to access the certificate that is returned by the CA.

Note: To receive a certificate by using the wsadmin tool, use the **receiveCertificate** command of the AdminTask object. For more information, see the PersonalCertificateCommands command group for the AdminTask object article.

About this task

WebSphere Application Server can receive only those certificates that are generated by a WebSphere Application Server certificate request. It cannot receive certificates that are created with certificate requests from other keystore tools, such as **iKeyman** and **keyTool**.

Complete the following steps in the administrative console:

Procedure

1. Click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > *ssl_configuration* > Key stores and certificates >** [*keystore*].
2. Under Additional Properties, click **Personal certificates**.
3. Select a personal certificate.
4. Click **Receive a certificate from a certificate authority**.
5. Type the full path and name of the certificate file.
6. Select a data type from the list.
7. Click **Apply** and **Save**.

Results

The keystore contains a new personal certificate that is issued by a CA. The original certificate request is changed to a personal certificate.

What to do next

The SSL configuration is ready to use the new CA-signed personal certificate.

Export certificate to a keystore file or a managed keystore

Use this page to specify a personal certificate to export to a keystore file or a managed keystore.

To view this administrative console page, click **Security > SSL certificate and key management**. Under Configuration settings, click **Manage endpoint security configurations > {Inbound | Outbound} > *ssl_configuration***. Under Related items, click **Key stores and certificates > *keystore*** . Under Additional Properties, click **Personal certificates**. Select a personal certificate using the check box. Then click the **Export** button.

Certificate alias to export:

Displays the name of the certificate that you selected to export on the previous panel.

Information	Value
Data type:	Text

Keystore Password:

Type in the password of the keystore to use for the export.

Information	Value
Data type:	Text

Alias:

Specifies the alias that the personal certificate is referenced by in the destination keystore.

Information	Value
Data type:	Text

Managed key store:

Select this option with the radio button. Then select a keystore from the pull-down list, which is managed by the security configuration, to export the certificate to.

Information	Value
Data type:	Drop-down list

Key file name:

Select this option with the radio button. Then type the keystore file name into which the exported certificate is added. If the keystore file name already exists, the exported certificate is added. If the keystore file name does not already exist, a keystore file name is created, and the exported certificate is added.

Information	Value
Data type:	Text

Type:

Specifies the type of keystore file. The valid types are listed in the drop-down list.

Information	Value
Data type:	Text
Default:	PKCS12

Key file password:

Specifies the password that is used to access the keystore file.

Information	Value
Data type:	Text

Import certificate from a key file or managed keystore

Use this page to specify a personal certificate to import from a keystore or key file.

To view this administrative console page, click **Security > SSL certificate and key management**. Under Configuration settings, click **Manage endpoint security configurations > {Inbound | Outbound} > *ssl_configuration***. Under Related items, click **Key stores and certificates > *keystore*** . Under Additional Properties, click **Personal certificates**. Select a personal certificate using the check box. Then click the **Import** button.

Managed key store:

Select this option with the radio button. This selection indicates that the keystore that contains the certificate to import is a managed keystore.

Information	Value
Data type:	radio button

Get key store aliases:

Clicking this button queries the configuration for the list of keystore aliases for which the certificate will be imported to.

Key store:

Select an alias of the keystore from the pull-down list of managed keystores that are managed by the security configuration. The alias you select identifies the keystore that contains the certificate to import.

Information	Value
Data type:	drop-down list

Key store password:

Specifies the password for the keystore to use for import.

Information	Value
Data type:	Text

Key store file:

Select this option with the radio button. This selection indicates a keystore file that contains the certificate to import.

Information	Value
Data type:	radio button

Key file name:

Specifies the fully qualified path to keystore file that contains the certificate to import.

Information	Value
Data type:	Text

Get key file aliases:

Clicking this button, queries the key file for the aliases of all the personal certificates in the keystore from which to choose.

Type:

Specify the type of keystore file. Select a valid type from the drop-down list.

Information	Value
Data type:	Text

Key file password:

Type the password that is used to access the keystore file.

Information	Value
Data type:	Text

Certificate alias to import:

Specifies the certificate alias identified as the **Key file name** that you want to import into the current keystore.

Information	Value
Data type:	Text
Default:	(none)

Imported certificate alias:

Specifies the new alias that you want the certificate to be named in the current keystore.

Information	Value
Data type:	Text

Receive certificate from CA

Use this page to import your personal certificate from the certificate authority (CA). The imported certificate replaces the temporary certificate associated with the public/private keys in the certificate request that is stored in the key store.

To view this administrative console page, click **Security > SSL certificate and key management**. Under Configuration settings, click **Manage endpoint security configurations > {Inbound | Outbound} > *ssl_configuration***. Under Related items click **Key stores and certificates > *key store*** . Under Additional Properties, click **Personal certificates > Receive certificate from certificate authority**.

Certificate file name:

Specifies the filename that contains the certificate generated by the certificate authority (CA).

Information	Value
Data type:	Text

Data type:

Specifies the format of the file that is either Base64 encoded ASCII data or Binary DER data.

Information	Value
Data type:	Text
Default:	Base64-encoded ASCII data

Replace a certificate

Use this page to specify two certificates: the first selected certificate is replaced by the second selected certificate. The replace function replaces all the old signer certificates in key stores that are managed throughout the cell with the new signer from the new certificate. The same level of trust that was established with the old certificate is maintained. All places the certificate's alias is referenced in the security configuration will be replaced with the certificate's alias. The alias could be referenced on a security object like the SSL configuration, the dynamic outbound endpoint SSL configuration and key set groups.

To view this administrative console page, click **Security > SSL certificate and key management**. Under Configuration settings, click **Manage endpoint security configurations > {Inbound | Outbound} > *ssl_configuration***. Under Related items, click **Key stores and certificates > *key store*** . Under Additional Properties click **Personal certificates**. Select a personal certificate, then click the **Replace** button.

Old certificate
Specifies the certificate that you want to replace.

Information	Value
Data type:	Text

Replace with
Specifies the certificate that you want to replace the old certificate.

Information	Value
Data type:	Text
Default:	(none)

Delete old certificate after replacement
Specifies that you want to delete the old certificate and all associated signer certificates after the new certificate replaces it. If you do not replace the old personal certificate, it might be assigned a new alias name.

Information	Value
Default:	Disabled

Delete old signers
Specifies that you want to delete the old signer certificates that are associated with the old certificate after the new signer certificates replace them. If you do not delete the old signer certificates, they might be assigned new alias names.

Information	Value
Default:	Disabled

Extracting a signer certificate from a personal certificate

Personal certificates contain a private key and a public key. You can extract the public key, called the *signer certificate*, to a file, then import the certificate into another keystore. The client requires the signer portion of a personal certificate for Security Socket Layer (SSL) communication.

Before you begin

The keystore that contains a personal certificate must already exist.

Alternative Method: To extract a signer certificate from a personal certificate using the wsadmin tool, use the **extractCertificate** command of the AdminTask object. For more information, see the PersonalCertificateCommands command group for the AdminTask object article.

About this task

Complete the following steps in the administrative console:

Procedure

1. Click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > *ssl_configuration* > Key stores and certificates > *keystore*** .
2. Under Additional Properties, click **Personal certificates**.
3. Select a personal certificate.
4. Click **Extract**.
5. Type the full path for the certificate file name. The signer certificate is written to this certificate file.
6. Select a data type from the list.
7. Click **Apply**.

Results

The signer portion of the personal certificate is stored in the file that is provided.

What to do next

This signer can now be imported into other keystores.

Extract certificate

Use this page to extract the signer from the personal certificate and store it in a file. The certificate can be added to a trust store for trust verification. When extracting the signer from a chained personal certificate, the signer at the top level of the chain is extracted.

To view this administrative console page, click **Security > SSL certificate and key management**. Under Configuration settings, click **Manage endpoint security configurations > {Inbound | Outbound} > *ssl_configuration***. Under Related items, click **Key stores and certificates > *key store*** . Under Additional Properties, click **Personal certificates > Extract**.

Certificate alias to extract

Displays the name of the certificate that you selected for extraction on the previous panel.

Information	Value
Data type:	Text

Certificate file name

Specifies the fully qualified path where the certificate file will reside.

Information	Value
Data type:	Text

Data type

Specifies the format of the file, which is either Base64-encoded ASCII data or Binary DER data.

Information	Value
Data type:	Text
Default:	Base64-encoded ASCII data

Extract signer certificate

Use this page to extract a signer certificate from the keystore to a file so that it can be added elsewhere.

To view this administrative console page, click **Security > SSL certificate and key management**. Under Configuration settings, click **Manage endpoint security configurations > {Inbound | Outbound} >**

ssl_configuration. Under Related items, click **Key stores and certificates** > ***key store***. Under Additional Properties, click **Signer certificates** . Select a signer certificate, then click the **Extract** button

File name
Specifies the fully qualified file name where the extracted signer certificate is placed.

Information	Value
Data type:	Text

Data type
Specifies the format of the file, which is either Base64 encoded ASCII data or Binary DER data.

Information	Value
Data type:	Text

Retrieving signers using the retrieveSigners utility at the client
The client requires the signer certificates from the server to be able to communicate with WebSphere Application Server. Use the **retrieveSigners** command to get the signer certificate from a server.

Before you begin

The retrieveSigners utility is located in one of the following directories, depending on your operating system:

- Windows *profile_root*\bin . For example: `C:\WebSphere\AppServer\profiles\AppSrv01\bin`
- AIX HP-UX Solaris ../*profile_root*/bin

In this release, a Java client that does not have access to a stdin console prompt should use the retrieveSigners utility to download the signers from the remote server key store when signers are needed for a Secure Sockets Layer (SSL) handshake. For example, you might interpret the client as failing to respond if an applet client or Java Web Start Client application cannot access the stdin signer exchange prompt. Thus, you must add the WebSphere Java method call **com.ibm.wsspi.ssl.RetrieveSignersHelper.callRetrieveSigners** to your client application to retrieve the signers and to avoid running the retrieveSigners utility manually.

Use the retrieveSigners utility for situations where you cannot verify whether or not the com.ibm.ssl.enableSignerExchangePrompt= property is enabled or disabled when the application makes a request. Set the com.ibm.ssl.enableSignerExchangePrompt= property to `false` in the ssl.client.props file if you cannot see the console.

Alternatively, you can manually create the server key in the client truststore.

About this task

Complete the following steps, as required:

Procedure
1. Use the **retrieveSigners** command to get the signer certificate from a server. You can find details about the **retrieveSigners** parameters in “Secure installation for client signer retrieval in SSL” on page 698.
2. If the client and server are on the same machine, you will need only the *remoteKeyStoreName* and *localKeyStoreName* parameters. The most typical key store to reference on a remote system is `CellDefaultTrustStore` on a network deployed environment and `NodeDefaultTrustStore` on an application server.

3. When retrieving signers from a remote server, add these required connection-related parameters: **–host** *host*, **–port** *port*, **–conntype** {*RMI* | *SOAP*}.
4. Use the **–autoAcceptBootstrapSigner** parameter if you want to enable automation of the signer retrieval. This parameter automatically adds to the server all the signers that are needed to make the connection.

Results

After running, the command displays the SHI-1 digest of the signers added. The output looks similar to the following output:

```
C:\WebSphere\AppServer\profiles\AppSrv01\bin\retrieveSigners.bat
CellDefaultTrustStore ClientDefaultTrustStore

CWPKI0308I: Adding signer alias "default_signer" to local keystore
          "ClientDefaultTrustStore" with the following SHA digest:
```

Example

The following examples illustrate how to call the `retrieveSigners.bat` file.

To retrieve signers on the same system, enter:

```
profile_root\bin\retrieveSigners.bat CellDefaultTrustStore ClientDefaultTrustStore
```

To retrieve signers on a remote system with a SOAP connection, enter:

```
profile_root\bin\retrieveSigners.bat CellDefaultTrustStore ClientDefaultTrustStore
-host myRemoteHost -port 8879 -conntype SOAP -autoAcceptBootstrapSigner
```

To retrieve signers on a remote system with an RMI connection, enter:

```
profile_root\bin\retrieveSigners.bat CellDefaultTrustStore ClientDefaultTrustStore
-host myRemoteHost -port 2809 -conntype RMI -autoAcceptBootstrapSigner
```

To retrieve signers on a remote system that has security enabled, enter:

```
profile_root\bin\retrieveSigners.bat CellDefaultTrustStore ClientDefaultTrustStore
-host myRemoteHost -port 8879 -conntype SOAP -user testuser -password testuserpwd
-autoAcceptBootstrapSigner
```

Changing the signer auto-exchange prompt at the client

For clients to communicate with WebSphere Application Server, clients must obtain a signer certificate from the server. Clients can use the **retrieveSigners** command to connect to a server to obtain the appropriate signer. A prompt displays that asks whether or not you want to add a signer to the truststore. If the Secure Sockets Layer (SSL) configuration uses an automated script that might hang, use the prompt to obtain the certificate.

Before you begin

The com.ibm.ssl.enableSignerExchangePrompt property in the *profile_home*/properties/ssl.client.props file controls the signer certificate prompt. By default, this property is set to `true`, meaning the prompt is enabled.

About this task

Complete the following steps to disable or enable the signer-exchange prompt at the client:

Procedure

1. Open the *profile_home*/properties/ssl.client.props file using an editor.
2. Locate the section containing the SSL configuration information for the client that you are working with.
3. Change the value of the com.ibm.ssl.enableSignerExchangePrompt property to `false` if you do not want the signer-exchange prompt, or set it to `true` if you want to be prompted.

4. Save and close the file.

Results

When the com.ibm.ssl.enableSignerExchangePrompt property is set to `false`, no prompt displays if a signer is not trusted. In this case the SSL handshake fails. Once the proper signer for the connection being made is manually installed in the trust store, the SSL handshake can succeed.

When the com.ibm.ssl.enableSignerExchangePrompt property is set to `gui` or `true`, a signer-exchange window is displayed, and you are asked to accept or reject the certificate. If you accept the certificate, it is installed in the trust store automatically and the handshake succeeds. If you reject the certificate, it does not get installed in the trust store and the handshake fails since the certificate is not trusted.

When the com.ibm.ssl.enableSignerExchangePrompt property is set to `stdin`, a signer-exchange ASCII prompt is displayed, and you are asked to accept or reject the certificate. If you accept the certificate, it is installed in the trust store automatically and the handshake succeeds. If you reject the certificate, it does not get installed in the trust store and the handshake fails since the certificate is not trusted.

The prompt looks like the following example:

Example

```
C:\WebSphere\AppServer\profiles\dmgr\bin>serverStatus -all
ADMU0116I: Tool information is being logged in file
           C:\WebSphere\AppServer\profiles\Dmgr\logs\serverStatus.log
ADMU0128I: Starting tool with the dmgr profile
ADMU0503I: Retrieving server status for all servers
ADMU0505I: Servers found in configuration:
ADMU0506I: Server name: dmgr

*** SSL SIGNER EXCHANGE PROMPT ***
SSL signer from target host 192.174.1.5 is not found in truststore
C:/WebSphere/AppServer/profiles/Dmgr/etc/trust.p12.
```

Verify that the digest value matches what is displayed at the server in the following signer information:

```
Subject DN:    CN=hostname.austin.ibm.com, O=IBM, C=US
Issuer DN:     CN=hostname.austin.ibm.com, O=IBM, C=US
Serial number: 1128544457
Expires:       Thu Oct 20 15:34:17 CDT 2006
SHA-1 Digest:  91:A1:A9:2D:F2:7D:70:0F:04:06:73:A3:B4:A4:9C:56:9D:A8:A3:BA
MD5 Digest:    88:72:C5:88:00:1C:A7:FA:D6:EB:04:88:AC:A1:C9:13

Add signer to the truststore now? (y/n) y
A retry of the request might need to occur.
ADMU0508I: The Application Server "server1" is STARTED.
```

What to do next

Clients can instigate communications for various processes using signer certificates obtained from WebSphere Application Server.

Retrieving signers from a remote SSL port

To perform Secure Sockets Layer (SSL) communication with a server, WebSphere Application Server must retrieve a signer certificate from a secure remote SSL port during the handshake. After the signer certificate is retrieved, you can add the signer certificate to a keystore.

Before you begin

The keystore that is to contain the signer certificate must already exist.

Alternative Method: To retrieve a signer certificate from a port using the wsadmin tool, use the **retrieveSignerFromPort** command of the AdminTask object. For more information, see the SignerCertificateCommands command group for the AdminTask object article.

About this task

Complete the following steps in the administrative console:

Procedure

1. Click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > Key stores and certificates > *keystore* > Signer certificates > Retrieve from port**.
2. Click **Retrieve from port**.
3. Type the host name of the machine on which the signer resides.
4. Type the port location on the host machine on which the signer resides. The port location is not limited to ports on WebSphere Application Server. The ports can include Lightweight Directory Access Protocol (LDAP) ports or ports on any server on which an SSL port is already configured, such as `SIB_ENDPOINT_SECURE_ADDRESS`.
5. Select an SSL configuration for the outbound connection from the list.
6. Type an alias name for the certificate.
7. Click **Retrieve signer information**. A message window displays information about the retrieved signer certificate, such as: the serial number, issued-to and issued-by identities, SHA hash, and expiration date. If a chained certificate is on the port, information about the root is displayed.
8. Click **Apply**. This action indicates that you accept the credentials of the signer.

Results

The signer certificate that is retrieved from the remote port is stored in the keystore.

What to do next

An SSL configuration or client process that requires an SSL connection to the server can use the retrieved and approved signer certificate.

Retrieve from port

Use this page to retrieve a signer certificate from a remote SSL port. The system connects to the specified remote SSL host and port and receives the signer during the handshake using an SSL configuration.

To view this administrative console page, click **Security > SSL certificate and key management**. Under Configuration settings, click **Manage endpoint security configurations > {Inbound | Outbound} > *ssl_configuration***. Under Related items, click **Key stores and certificates > *key store***. Under Additional Properties, click **Signer certificates**. Then click the **Retrieve from port** button.

To retrieve a signer certificate from a specific port, you enter the host and port, select an SSL configuration from the pull-down list, and enter an alias to identify the signer certificate. Click **Retrieve Signer Information** and information about the signer certificate is displayed, such as the serial number of the certificate, who the certificate is issued to and by, the certificate finger print, and the expiration information for the certificate. If you want the certificate to be stored in the keystore, click **Apply** or **Save**.

Host

Specifies the host name to which you connect when attempting to retrieve the signer certificate from the Secure Sockets Layer (SSL) port.

Information	Value
Data type:	Text

Port
Specifies the SSL port to which you connect when attempting to retrieve the signer certificate.

Information	Value
Data type:	Text

SSL configuration for outbound connection
Specifies the SSL configuration that is used to connect to the previously specified SSL port. This configuration is also the SSL configuration that contains the signer after retrieval. This SSL configuration does not need to have the trusted certificate for the SSL port as it is retrieved during validation and presented here.

Information	Value
Data type:	Text

Alias
Specifies the certificate alias name that you want to reference the signer in the key store, which is specified in the SSL configuration.

Information	Value
Data type:	Text

Retrieved signer information
Specifies the signer certificate information if it is retrieved from the remote host and port.

Adding a signer certificate to a keystore

Signer certificates establish the trust relationship in SSL communication. You can extract the signer part of a personal certificate from a keystore, and then you can add the signer certificate to other keystores.

Before you begin

The keystore that you want to add the signer certificate to must already exist.

Alternative Method: To add a signer certificate to a keystore by using the wsadmin tool, use the **addSignerCertificate** command of the AdminTask object. For more information, see the SignerCertificateCommands command group for the AdminTask object article.

Note: If the security custom property com.ibm.websphere.security.OverwriteAndReplaceOnImport is set to `true` then import certificate imports a certificate and overwrites an existing certificate. It then perform the certificate replace operation on that certificate. Typically, an existing certificate cannot be overwritten by a certificate that is being imported. The task also replaces all signer certificates from the original certificate and replaces them with the signer certificate from the new certificate that is being imported

About this task

Complete the following steps in the administrative console:

Procedure

1. Click **Security > SSL certificate and key management > Manage endpoint security configurations > Inbound | Outbound > *ssl_configuration* > Key stores and certificates**.
2. Select a ***keystore*** from the list of keystores.
3. Click **Signer certificates**.
4. Click **Add**.
5. Enter an alias for the signer certificate in the **Alias** field
6. Enter the full path to the signer certificate file in the **File name** field.
7. Select a data type from the list in the **Data type** field.
8. Click **Apply**.

Results

When these steps are completed, the signer from the certificate file is stored in the keystore. You can see the signer in the keystore files list of signer certificates. Use the keystore to establish trust relationships for the SSL configurations.

Add signer certificate settings

Use this page to add a signer certificate in a certificate file to the keystore in the security configuration.

To view this administrative console page, click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > *ssl_configuration* > Key stores and certificates > *keystore* > Signer certificates > Add**.

Alias

Specifies the alias that is used to identify the signer certificate in the keystore.

Information	Value
Data type:	String

File name

Specifies the path to the filename where the signer certificate is located.

Information	Value
Data type:	String

Data type

Specifies the format of the file, which is either Base64 encoded ASCII data or Binary DER data.

Information	Value
Data type:	String

Signer certificates collection

Use this page to manage signer certificates in key stores. Signer certificates are used by Java Secure Socket Extensions (JSSE) to validate certificates sent by the remote side of the connection during a Secure Sockets Layer (SSL) handshake. If a signer does not exist in the trust store that can validate the certificate sent, the handshake fails and generates an "unknown certificate" error.

To view this administrative console page, click **Security > SSL certificate and key management**. Under Configuration settings, click **Manage endpoint security configurations > {Inbound | Outbound} > *ssl_configuration***. Under Related items, click **Key stores and certificates > *key store***. Under Additional Properties, click **Signer certificates**.

Table 107. Signer certificates buttons. This table lists the signer certificates buttons.

Button	Resulting action
Add	Adds a new trusted (signer) certificate.
Delete	Deletes an existing signer certificate.
Extract	Extracts a signer certificate from a personal certificate to a file.
Retrieve from port	Makes a test connection to an SSL port and retrieves the signer from the server during the handshake. The information from the certificate will be displayed so you can decide whether to trust it based upon the MD5 and/or SHA hash.

Alias
Specifies the alias for this signer certificate in the key store.

Issued to
Specifies the distinguished name of the entity that requested the certificate.

Fingerprint (SHA digest)
Specifies the Secure Hash Algorithm (SHA hash) of the certificate. This can be used to verify the hash for the certificate at another location, such as the client side of a connection.

Expiration
Specifies the expiration date of the signer certificate for validation purposes.

Signer certificate settings
Use this page to verify the general properties of the selected signer certificate.

To view this administrative console page, click **Security > SSL certificate and key management**. Under Configuration settings, click **Manage endpoint security configurations > {Inbound | Outbound} > *ssl_configuration***. Under Related items, click **Key stores and certificates > *key store***. Under Additional Properties, click **Signer certificates**. Then click on a signer certificate.

Alias
Specifies the alias for this signer certificate in the key store.

Version
Specifies the version of the personal certificate. Valid versions include X509 V3, X509 V2, or X509 V1.

Key size
Specifies the key size of the public key used by the signer certificate.

Serial number
Specifies the certificate serial number that is generated by the issuer of the certificate.

Validity period
Specifies the begin and end dates of the certificate.

Issued to
Specifies the distinguished name of the entity that requested the certificate.

Issued by
Specifies the distinguished name of the entity that issued the certificate. This name is the same as the issued-to distinguished name when the signer certificate is self-signed.

Fingerprint (SHA Digest)
Specifies the Secure Hash Algorithm (SHA) hash of the certificate, which can be used to verify the hash for the certificate at another location such as the client side of a connection.

Signature algorithm
Specifies the algorithm that is used to sign the certificate.

Adding a signer certificate to the default signers keystore

Signer certificates are added to a keystore on the client side of an SSL communication to establish trust with the server. There is common practice for keystores to have trust established when they are created. The **DmgrDefaultSignersStore** on a deployment manager and the **NodeDefaultSignersStore** on a stand alone application server are created to hold signer certificates used to establish trust by default in newly create keystores.

Before you begin

The default signers key store is created during profile creation and contains the signer certificate of the server default root certificate. Additional signer certificates can be added to the default signers key store at any time. Anytime a keystore is created using the admin console or by using the **createKeyStore** AdminTask object in scripting, all signer certificates from the default signer store are added to the newly created keystore.

Alternative Method:

- To add a signer certificate to a default signer keystore by using the wsadmin tool, use the **addSignerCertificate** command of the AdminTask object.
- To create a new keystore by using the wsadmin tool, use the **createKeyStore** command of the AdminTask object.
- To extract the signer from a personal certificate using the wsadmin tool, use the **extractCertificate** of the AdminTask object.
- To exchange a signer certificate using the wsadmin tool, use the **KeyStoreCommands** command group for the AdminTask object.

For more information, see the SignerCertificateCommands command group for the AdminTask object article and the KeyStoreCommands command group for the AdminTask object article.

Procedure

1. If the certificate is in a certificate file, it can be added to the default signer keystore using the administrative console.
   a. Click **Security > SSL certificate and key management**.
   b. Under Related Items, click **Key stores and certificates**.
   c. c. Select Default signers keystore under KeyStore Usages. A panel displaying a list of keystores appears.
   d. Click on **DmgrDefaultSignersStore**.
   e. Under Additional Properties, click **Signer certificates**.
   f. Click **Add** .
   g. Enter an alias in the alias box, a path to the certificate file in the filename box, and an asterisk (•). Select the format of the certificate file from the pull down list in the “Data type” box.
   h. Click **Apply** then **Save**.

   Note: You can also perform this addition using the AdminTask, **addSignerCertificate**.

2. If the signer certificate form of a personal certificate needs to be added to default signers keystore, you can extract the signer from the personal certificate to a certificate file or the signer can be extracted directly to the default signers keystore. To extract a signer certificate from a personal certificate to a certificate file,
   a. Click **Security > SSL certificate and key management**.
   b. Under Related Items, click **Key stores and certificates**.
   c. c. Select All under Keystore Usages. A panel displaying a list of keystores appears.
   d. Click on the keystore name
   e. Under Additional Properties, click **Personal certificates**.
   f. Select a personal certificate.
   g. Click **Extract**.
   h. Enter the path to the certificate file in “Certificate file name” box and select a format type from the pull down list in “Data type” box
   i. Click **Apply** then **Save**.
   j. The signer can be added to the default signers keystore by following step 1.

   Note: You can also extract the signer from a personal certificate using scripting and the AdminTask **extractCertificate**.
3. To extract a signer certificate to the default signers keystore, an exchange of the signer certificate can be performed from the administrative console.
   a. Click **Security > SSL certificate and key management**
   b. Under Related Items, click **Key stores and certificates**.
   c. c. Select All under Keystore Usages. A panel displaying a list of keystores appears.
   d. Click on the default signers keystore and the keystore that contains the personal certificate whose signer certificate is needed.
   e. Click **Exchange Signers**.
   f. Select the personal certificate whose signer is needed.
   g. Click **Add**.
   h. Click **Apply** then **Save**.

   Note: You can also perform the exchange using the AdminTask, **exchangeSigner**.

   Note: A DataPower certificate can be removed from the default signers keystore if it is present. If you are not using the DataPower appliance manager you should remove the DataPower certificate from the default trust store to avoid unintentional trust relationships. However, if you start to use DataPower appliance manager at a later date you must add the DataPower certificate back to the default trust store.

Results

When these steps are completed, the signer from the certificate file is stored in the default signers keystore. You can see the signer in the keystore files list of signer certificates.

What to do next

The new keystore will contain the default signers that were added to the default signers keystore.

Exchanging signer certificates

To establish trust relationships, you can exchange signer certificates between keystores. When you exchange signer certificates, you are extracting a personal certificate from one keystore and adding it to another keystore as a *signer certificate*.

Before you begin

To exchange signer certificates, there must be two keystores.

About this task

Complete the following steps in the administrative console:

Procedure

1. Click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > *ssl_configuration* > Key stores and certificates**.
2. Select two keystores from the list of keystores.
3. Click **Exchange signers**.
4. Select any of the certificates in the first personal certificates list, and click **Add**. After adding, the signer part of the selected personal certificate appears in the other (second) keystore signers list.
5. Select any of the certificates in the second personal certificates list, and click **Add**. After adding, the signer part of the selected personal certificate appears in the other (first) keystore signers list.
6. Optional: If you need to remove any of the certificates from either of the signers lists, highlight one or more of the certificates, and click **Remove**.
7. Click **Apply** and **Save**.

Results

The signer certificate appears in the list for each keystore.

What to do next

The extracted signer certificate is available to both keystores during the connection handshake.

Keystores and certificates exchange signers

Use this page to extract the signer part of a personal certificate from one keystore and add it to another keystore as a signer certificate. Signer certificates can also be listed, and they will be added to the other keystore as well.

To view this administrative console page, click **Security > SSL certificate and key management**. Under Configuration settings, click **Manage endpoint security configurations > {Inbound | Outbound} > *ssl_configuration***. Under Related items, click **Key stores and certificates** then select two key stores to exchange and click the **Exchange signers**.

Note: Any changes made to this panel are permanent.

[keystore] personal certificates

Specifies the personal certificates and signer certificates that are currently stored in the specified keystore.

Press and hold the **Ctrl** key to select more than one item from the list.

Information	Value
Data type:	Text

[keystore] signers
Specifies the trusted signer certificates that are currently stored in the specified keystore and selected for the exchange.

Press and hold the **Ctrl** key to select more than one item from the list.

Information	Value
Data type:	Text

Add
Specifies to extract the signer from the selected personal certificate in the keystore list on the left and add it to the signers list of the keystore on the right.

After the certificate is added, it no longer displays in the left-hand list. The personal certificate is still in the keystore, but it is no longer selectable

Remove
Specifies to remove a selected signer from the signers list of the keystore on the right. The removed certificate displays in the keystore list on the left.

Configuring certificate expiration monitoring

When certificates expire, they can no longer be used by the system. WebSphere Application Server provides a utility to monitor certificates that are close to expiration or have already expired. You can schedule certificate monitoring, or you can request certificate monitoring on demand. You can also configure options for deleting expired certificates and for recreating certificates.

Before you begin

Important: The Certificate Expiration Monitor does not handle replacing client self-signed certificates and is not capable of sending the new signer certificate needed for trust. If the client is a web server plug-in, it will not be able to securely communicate with the application server after self-signed certificate replacement.

WebSphere Application Server notifies you when a certificate is about to expire. Complete the information required for notification messaging in “Notifications” on page 805.

About this task

Complete the following configuration steps in the administrative console:

Procedure
1. Click **Security > SSL certificate and key management > Manage certificate expiration**.
2. Type a number for the number of days threshold in the **Expiration notification threshold** field. WebSphere Application Server issues an expiration warning *n* number of days before expiration.
3. Select or check one or more of the following options:
   - **Expiration check notification**. Select the method from the list that you want to use to receive your notification.
   - **Automatically replace expiring self-signed certificates**. If you do not want to recreate the self-signed certificate, clear the check box.

Attention: When using writable System Authorization Facility (SAF) keyrings in your configuration, the certificate expiration monitor does not replace expired certificates in the writable SAF keyrings, but only provides a notification of the expiration.

- **Delete expiring certificates and signers after replacement**. If you do not want to delete the expired certificates and signers, clear the check box.
- **Enable checking**. If you do not want to have certificate monitoring enabled, clear the check box.

4. Enter the time of day when you want certificate monitoring to take place to schedule the running of the certificate expiration monitor.
5. Select one of the following options:
   - **Check by calendar**. For **Weekday**, enter the day of week that you want to run the certificate expiration monitor. For **Repeat Interval**, specify the frequency to run the certificate monitor.
   - **Check by number of days**. Enter a number for how frequently the monitor runs, in number of days.
6. Type the number of days before the threshold date in which the certificate monitor warns that a certificate is about to be replaced. When a certificate is within the expiration threshold, and automatic replacement is enabled, certificates are replaced. This value specifies the time period before the threshold when warnings are issued by the certificate monitor concerning upcoming replacement dates.
7. Click **Apply**.

Results

After completing the settings, a certificate expiration monitor object and a schedule are set up in the configuration. The certificate expiration monitor runs according to the configurations options that you configured.

What to do next

You can generate reports that state which certificates have expired. The reports identify the notifications of certificate replacements and deletions. The report is sent according to the notification option that you specified.

Manage certificate expiration settings

Use this page to configure the certificate expiration monitor.

To view this administrative console page, click **Security > SSL certificate and key management**. Under Configuration settings, click **Manage certificate expiration**.

Attention: To see the changes to the Expiration checking fields, you must click **Apply**.

Start now

Specifies to start certificate monitoring. When the monitor runs, it visits all the key stores and checks to see if they are within certificate expiration range. If you set the option to delete or replace expired certificates, you can run these operations immediately by pressing **Start now**.

Expiration notification threshold

Specifies the period of time that occurs chronologically just before the expiration day of the certificate, within which, if the ExpirationMonitor thread runs, and **Automatically replace expiring self-signed and chained certificates** is enabled, a new self-signed or chained certificate is generated. By default, the replacement period for the certificate is 60 days in length or less as defined in the daysBeforeNotification property.

There is a pre-notification period where the certificate is added to the notification list but not touched for 90 days prior to the 60 days. By default, this pre-notification period is 90 days in length as defined in the com.ibm.ws.security.expirationMonitorNotificationPeriod property.

Information	Value
Data type:	Integer
Default:	60 days or less

Enable checking
Specifies the certificate monitor is active and will run as scheduled.

Scheduled time of day to check for expired certificates
Specifies the scheduled time that the system checks for expired certificates.

You can type the scheduled time in hours and minutes, specify either A.M. or P.M., or 24-hour.

Information	Value
Data type	Integer
Default:	0, 0
Range:	1–12, 0–59

Check by calendar
Indicates that you want to schedule a specific day of the week on which the expiration monitor runs. For example, it might run on Sunday.

Information	Value
Default:	Enabled

Weekday
Specifies the day of the week on which the expiration monitor runs if **Check on a specific day** is selected.

Information	Value
Default:	Sunday
Range:	Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday

Repeat interval
Specifies the period of time between each schedule time to check for expired certificates or the interval between schedule checks.

Information	Value
Default:	Daily
Range:	Daily, Weekly

Check by number of days
Specifies that you want to schedule a specific number of days between each run of the expiration monitor. The day of the week on which this occurs is not counted. For example, if you set the interval to check for expired certificates every seven days, the expiration monitor runs on day eight.

Information	Value
Default:	Disabled

Next start date
Specifies the date for the next scheduled check. This allows the deployment manager to be stopped and restarted without resetting the date.

Expiration check notification
Specifies the notification type (either email, or an entry in the system log) when an expiration monitor runs.

Information	Value
Default:	

Automatically replace expiring self-signed certificates and chained certificates
Specifies a new self-signed certificate or chained certificate be generated using the same certificate information if the expiration notification threshold is reached. The old certificate is replaced and uses the same alias. All old signers are managed by the key store configuration are also replaced. The system only replaces self-signed certificates.

Note: This checkbox is only applicable when using file based keystores.

Information	Value
Default:	Enabled

Delete expiring certificates and signers after replacement
Specifies whether to completely remove old, self-signed certificates from the key store during a replace operation or leave them there under a renamed alias. If an old certificate is not deleted, the system renames the alias so that the new certificate can use the old alias, which might be referenced elsewhere in the configuration.

Note: This checkbox is only applicable when using file based keystores.

Information	Value
Default:	Enabled

Notifications
Use this page to specify the generic notification definitions that are used in certificate expiration monitors.

To view this administrative console page, click **Security > SSL certificate and key management**. Under Configuration settings, click**Manage certificate expiration**. Under Related items, click **Notifications**.

Table 108. Notifications buttons. This table lists the notifications buttons.

Button	Resulting action
New	Adds a notification. The notification configures how the expiration monitor notifies the administrator of certificates that will expire within the specified threshold.
Delete	Deletes an existing notification.

Notification name
Specifies the notification name.

Message log
Specifies that this configuration intends to log certificate expiration information to the message log file.

Send Email
Specifies that this configuration intends to send certificate expiration information to the list of users in the email list.

List of email addresses
Specifies the email addresses that are sent notifications when certificates fall within the expiration threshold. You must specify the SMTP server for each email address. If an email address is not specified,

by default the application server assumes that the SMTP server is "smtp-server." For example, if you type *name*@domain, the SMTP server will be smtp-server.domain.

Notifications settings

Use this page to set properties for new notifications used in certificate expiration monitors or for security audit subsystem failures.

To view this administrative console page perform one of the following:

- Click **Security > SSL certificate and key management** > **Manage certificate expiration** >**Notifications > New**.
- Click **Security** > **Security auditing** > **Audit monitor** > **New**

.

Notification name

Specifies the name of the notification configuration.

Information	Value
Data type:	Text

Message log

Specifies that this configuration will log the notification to a message log file.

Information	Value
Default:	Disabled

Email sent to notification list

Specifies that this configuration send a notification as an email to the email list.

Information	Value
Default:	Disabled

Email address to add

Specifies the email addresses that are sent notifications. You must specify the SMTP server for each email address. If an email address is not specified, by default the application server assumes that the SMTP server is "smtp-server." For example, if you type *name*@domain, the SMTP server will be smtp-server.domain.

Information	Value
Data type:	Text (format as valid Internet mail address)

Add

Adds the email address to the right-hand list.

Remove

Removes the email address from the right-hand list.

Outgoing mail (SMTP) server

Specifies the SMTP server to be used with the email address. If none is specified, the email realm will be used.

Key management for cryptographic uses

WebSphere Application Server provides a framework for managing keys (secret keys or key pairs) that applications use to perform cryptographic operations on data. The key management framework provides an application programming interface (API) for retrieving these keys. Keys are managed in keystores so the keystore type can be supported by WebSphere Application Server, provided that the keystores can store the referenced key type. You can configure keys and scope keystores so that they are visible only to particular processes, nodes, clusters, and so on.

The key management infrastructure is based on two key configuration object types: key sets and key set groups. WebSphere Application Server uses a key set to manage instances of keys of the same type. You can configure a key set to generate a single key or a key pair, depending on the key or key pair generator class. A key set group manages one or more key sets and enables you to configure and generate different key types at the same time. For example, if your application needs both a secret key and key pair for cryptographic operations, you can configure two key sets, one for the key pair and one for the secret key that the key set group manages. The key set group controls the auto-generation characteristics of the keys, including the schedule. The framework can automatically generate keys on a scheduled basis, such as on a particular day of the week and time of day, so that key generation is done during off-peak hours.

Figure 1 shows an example of a key set group that is configured to manage two key sets: key set 1 and key set 2.

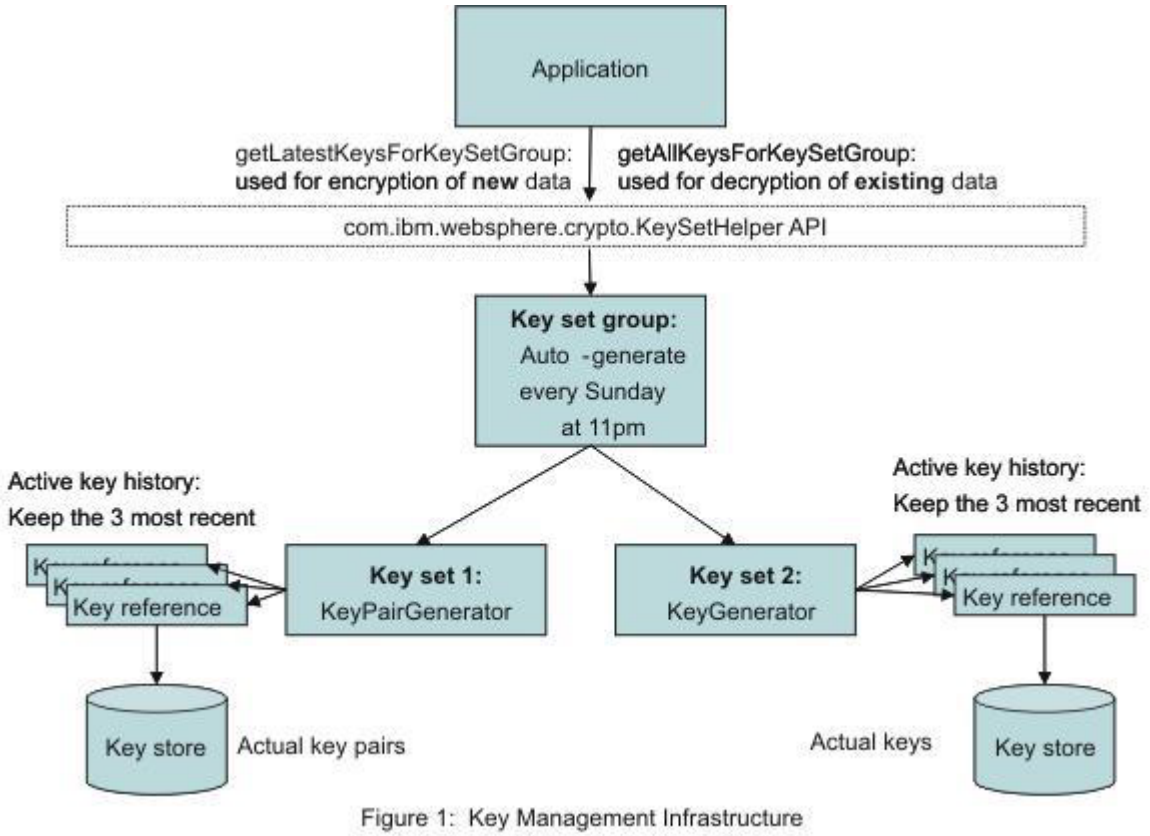


Figure 39.

Key set 1 generates key pairs. Key set 2 generates secret keys. The application needs both types of keys for its cryptographic operations, signing and encryption, on data. The keys for each key set need to be

generated in tandem. The application stores the key set group name with the encrypted data. The key set group generates a new set of keys every Sunday night at 11 P.M.. The application maintains key generation data for two weeks.

Creating a key set configuration

You can use key sets to manage multiple instances of cryptographic keys. WebSphere Application Server uses keys to encrypt or sign outbound data, and decrypt or verify inbound data during cryptographic operations.

Before you begin

You must have write-access to the keystore that will contain the keys after you generate them from a key set. However, if you want to generate keys outside of WebSphere Application Server, you can reference the keys from a read-only keystore that contains a secret key that you can access when you generate the keys. If you are creating a key pair using an X509Certificate and a PrivateKey object , see “Example: Developing a key or key pair generation class for automated key generation” on page 815.

About this task

Complete the following steps in the administrative console:

Procedure

1. Decide whether you want to create the key set at the cell scope or below the cell scope at the node, server, or cluster, for example:
   - To create a key set at the cell scope, click **Security > SSL certificate and key management > Key sets**.
   - To create a key set at a scope below the cell level, click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > *ssl_configuration* > Key sets**.
2. Click **New** to create a new key set.
3. Type a key set name. For example, `CellmyKey`.
4. Type a key alias prefix name. For example, `myKey`. This field specifies the prefix for the key alias when the new key is generated and stored in the keystore. Following the prefix is the key reference version number, for example, 2, so that the full key alias name would be `myKey_2`. If the key reference already has a specified alias for a key that exists in the keystore, then WebSphere Application Server ignores this field.
5. Type a key password. The key password protects the key in the keystore. This password is ignored by WebSphere Application Server if you already specified a password for the key alias reference. To check for a key reference password, click **Active key history** under Additional Properties. The key reference password protects keys that are generated by a key generator class.
6. Type the password again to confirm it.
7. Optional: Type the key generator class name. For example, `com.ibm.ws.security.ltpa.LTPAKeyGenerator`. The class name generates keys. If the class implements com.ibm.websphere.crypto.KeyGenerator, then a getKey method returns a java.security.Key object that is set in the keystore using the setKey method without a certificate chain. If the class implements com.ibm.websphere.crypto.KeyPairGenerator, then a getKeyPair method returns a com.ibm.websphere.crypto.KeyPair object that contains either a java.security.PublicKey and java.security.PrivateKey or a java.security.cert.Certificate and a java.security.PrivateKey object. The key generator class and the KeySetHelper API specify the details of the keys that are generated.
8. Optional: Select **Delete key references that are beyond the maximum number of keys** if you do not want old keys saved in the keystore after WebSphere Application Server removes their references

from the Active key history listing. The Active key history lists the keys that the KeySetHelper API is currently tracking. The number of keys in the list is equal to the number of keys that you specify in **Maximum number of keys referenced**.

9. Type a numeric value for the maximum number of keys referenced. For example, if you type `3` and select **Delete key references that are beyond the maximum number of keys**, the fourth key version generation automatically triggers WebSphere Application Server to delete the first key version from the keystore. If you choose not to delete the old keys, they do not display in the Active key history list but instead remain in the keystore where you can remove them manually.
10. Select a keystore from the drop-down list.
    - Select a JCEKS keystore if you are storing a secret key.
    - Select any keystore if you are storing a key pair with an X509Certificate and PrivateKey object.
11. Optional: Select **Generates key pair** if your key generator class name implements the com.ibm.websphere.crypto.KeyPairGenerator interface instead of the com.ibm.websphere.crypto.KeyGenerator interface. This option designates that the key references a key pair instead of a single key. A key pair contains both a public key and a private key. The WebSphere Application Server run time determines whether or not key pairs are stored and loaded differently than single keys.
12. Optional: Click **Apply** if you want to select **Active key history** under Additional Properties to add alias references or generate more keys.
    a. Click **Active key history**.
    b. Click **Add key alias reference** if you are not using the key generator class name to add key alias references to the keys that already exist in the keystore. Use this option to retrieve the keys from a read-only keystore without the key set generating them.
    c. Type an alias reference.
    d. Click **Generate key** if you want to generate a key using the class name that you defined in the key sets panel. Each new key increments numerically, for example, `myAlias_2`.
    e. Click **Apply**.
13. Click the key set name in the navigation path at the top of the panel.
14. Click **OK** and **Save**.

Results

You have created a key set that you can manage using the **Active key history** link. You can generate keys manually to associate them with specified key sets.

What to do next

After you generate new keys from a key set, you can access them programmatically using the com.ibm.websphere.crypto.KeySetHelper API. You must have Java 2 Security permissions, if enabled, to access keys in key sets. Specify the key set name within the fine-grained permissions, as in the following code sample: `WebSphereRuntimePermission "getKeySets.`*`keySetName`*`"`. For more information, see “Example: Retrieving the generated keys from a key set group” on page 814. To generate multiple key types at the same time or to schedule the key generation on a specific schedule, see “Creating a key set group configuration” on page 812.

Active key history collection

Use this page to manage key alias references.

To view this administrative console page, click **Security** > **SSL certificate and key management** > **Manage endpoint security configurations > {Inbound | Outbound}** > ***ssl_configuration*** > **Key Sets** > ***key set*** > **Active key history**.

Table 109. Active key history buttons. This table lists the active key history buttons.

Button	Resulting action
Add key alias reference	Adds a reference to a key that already exists in a key store. If a key generation class is configured, the references are added automatically during generation and do not need to be added manually.
Delete	Deletes an existing key reference. This action does not delete the key in the keystore.
Generate key	Generates a key. The button is displayed only if a generator class name is specified for the key set, and the selected key store is editable.

Alias reference
Specifies the name of the alias as it appears in the keystore.

Add key alias reference settings
Use this page to access key alias reference information.

To view this administrative console page, Under Configuration settings, click **Manage endpoint security configurations > {Inbound | Outbound} > *ssl_configuration***. Under Related items, click **Key Sets > *key set***. Under Additional Properties, click **Active key history** then click the **Add key alias reference** button.

Alias reference
Specifies the name of the alias as it appears in the key store.

Information	Value
Data type:	Text

Password
Specifies the key password to get access to the key. This password is enforced by the keystore for that specific key. If the key does not have a password, this field can be left blank.

Information	Value
Data type:	Text

Confirm password
Confirms the password entered in the previous field.

Information	Value
Data type:	Text

Key sets collection
Use this page to manage key sets, which control a set of key instances of the same type for use in cryptographic operations. The keys can either be generated using a custom class or reference keys that already exist in a keystore.

To view this administrative console page, click **Security > SSL certificate and key management**. Under Configuration settings, click **Manage endpoint security configurations > {Inbound | Outbound} > *ssl_configuration***. Under Related items, click **Key sets**.

Table 110. Key set buttons. This table lists the key set buttons.

Button	Resulting action
New	Adds a new key set.
Delete	Deletes an existing key set. Make sure the key set is not referenced by a key set group before deleting it.

Key set name
Specifies the key set name that is used to select the key set from a key set group and from runtime application programming interfaces (API).

Key store
Specifies the key store that contains the keys for storage, retrieval, or both.

Key alias prefix name
Specifies the prefix for the key alias when a new key is generated and stored in a key store. The rest of the key alias comes from the key reference version number.

For example, if the alias prefix is `mykey` and the key reference version is 2, the keystore references the key using alias `mykey_2`. If the key reference already has a specified alias for a key already existing in the keystore, this field is ignored.

Key sets settings
Use this page to set the properties for a new key set.

To view this administrative console page, click **Security** > **SSL certificate and key management**. Under Configuration settings, click**Manage endpoint security configurations** > **{Inbound | Outbound}** > ***ssl_configuration***. Under Related items, click **Key sets** > **New**.

Key set name
Specifies the key set name that is used to select the key set from a key set group and from runtime application programming interfaces (API).

Information	Value
Data type:	Text

Management scope
Specifies the scope where this Secure Sockets Layer (SSL) configuration is visible. For example, if you choose a specific node, then the configuration is only visible on that node and any servers that are part of that node.

Information	Value
Data type	List
Range:	Applicable scopes

Key alias prefix name
Specifies the prefix for the key alias when a new key is generated and stored in a keystore. The rest of the key alias comes from the key reference version number. For example, if the alias prefix is `mykey` and the key reference version is 2, the keystore references the key using alias `mykey_2`. If the key reference already has a specified alias for a key already existing in the keystore, this field is ignored.

Information	Value
Data type:	Text

Key password
Specifies the password used to protect the key in the keystore. If a password is specified in the key reference as well, this password is ignored. This password is used for keys that get generated by a key generator class.

Information	Value
Data type:	Text

Confirm password
Specifies the same password again to confirm it was entered correctly the first time.

Information	Value
Data type:	Text

Key generator class name
Specifies the class name that generates keys. If the class implements com.ibm.websphere.crypto.KeyGenerator, then a getKey() method should return a java.security.Key object that is set in the key store using the setKey method without a certificate chain. The key store type associated with the key set must support storing keys without certificates, such as JCEKS.

Information	Value
Data type:	Text

If the class implements com.ibm.websphere.crypto.KeyPairGenerator, then a getKeyPair() method should return a com.ibm.websphere.crypto.KeyPair object containing either a java.security.PublicKey and java.security.PrivateKey, or a java.security.cert.Certificate[] and a java.security.PrivateKey. The key generator class and the caller of the KeySetHelper API should know the details of the keys that are generated. This framework does not need to understand the key algorithms and lengths.

Delete key references that are beyond the maximum number of keys:

Specifies that the keys are deleted from the keystore at the same time that the key reference is deleted. The server deletes the older key references as the Maximum number of keys referenced value is exceeded.

Maximum number of keys referenced
Specifies the maximum number of key instances that are returned when keys from this key set are requested. The oldest key reference gets removed whenever a new key reference gets generated after the maximum has been reached.

Information	Value
Data type:	Integer
Default:	3

Key store
Specifies the key store that contains the keys for storage, retrieval, or both.

Information	Value
Data type:	Text

Generates key pair
Specifies that a key references a key pair instead of a key. The key pair contains both a public key and a private key.

Creating a key set group configuration

A key set group manages one or more key sets. WebSphere Application Server uses key set groups to automatically generate cryptographic keys or multiple synchronized key sets.

About this task

Complete the following steps in the administrative console:

Procedure

1. Decide whether you want to create the key set group at the cell scope or below the cell scope at the node, server, or cluster, for example.
   - To create a key set group at the cell scope, click **Security > SSL certificate and key management > Key set groups**.
   - To create a key set group at a scope below the cell level, click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > *SSL_configuration* > Key set groups**.
2. You can choose to generate a key for an existing key set group, delete an existing key set group, or create a new key set group.
   - To generate a key for an existing key set group, select a key set group from the list of existing key set groups, and click **Generate keys**. You have generated a new key for each key set in the selected group.
   - To delete an existing key set group, select a key set group from the list of existing key set groups, and click **Delete**. You have deleted the key set group.
   - To create a new key set group, go to step 3.

   CAUTION:
   Do not delete the cell or node LTPAKeySetGroup, which is used by the Lightweight Third Party Authentication (LPTA) mechanism.
3. Click **New** to create a new key set group.
4. Type a key set group name. You can reference this name by using the com.ibm.websphere.crypto.KeySetHelper API to retrieve the managed keys from an application.
5. Select one or more key sets from the Key sets list.

   Note: If the key set(s) you want is not listed, make sure that it was created at the same scope or a higher scope than where you are creating the new key set group.
6. Click **Add** to add the selected key set(s) to the new key set group.
7. Select **Automatically generate keys** to generate the new keys on a schedule. If you decide to generate keys automatically, then you must specify a scheduled time of day.
8. Specify the scheduled time to generate keys automatically in hours and minutes, A.M. or P.M., or every 24 hours.
9. You can choose to generate new keys on a specific day or at an interval.
   - Select **Generate on a specific day**. Select a day of the week from the drop-down list, and type a repeat interval number for the number of days between each key generation. This choice enables you to schedule key generation when your systems are least busy.
   - Select **Generate at an interval**. Type a repeat interval number for the number of days between each key generation. This choice enables you to schedule key generation more frequently than once a week.

   Note: The **Next start date** is a read-only field that specifies the date for the next scheduled generation. You can stop and restart the deployment manager or base application server without resetting this date. If you do not see the next start date appear after changing the configuration, click **OK** to save it, then check that the next start date displays.
10. Click **Save**.

Results

You have created a new key set group to manage key sets and key generation on a schedule.

What to do next

After you generate new keys from a key set, you can access them programmatically using the com.ibm.websphere.crypto.KeySetHelper API. You must have Java 2 Security permissions, if enabled, to access keys in key sets. Specify the key set name within the fine-grained permissions, as in the following code sample: `WebSphereRuntimePermission "getKeySets.`*`keySetName`*`"`. For more information, see “Example: Retrieving the generated keys from a key set group.”

Example: Retrieving the generated keys from a key set group

This example shows how applications can use the com.ibm.websphere.crypto.KeySetHelper API to retrieve managed keys from the KeySet or KeySetGroup configurations. Use the com.ibm.websphere.crypto.KeySetHelper API to get either the latest set of keys or all the keys in the KeySet or KeySetGroup object.

Use the latest keys when performing any new cryptographic operations. All of the other keys that are defined in the KeySet or KeySetGroup object are for the validation of previously performed cryptographic operations.

The following example uses a method that an application might use to initialize the keys in the associated KeySetGroup object. The application might want to store the keys in two separate maps, one for generation and one for validation. Refer to the API documentation for KeySetHelper API to determine which Java 2 Security requirements are required.

```
/**
     * Initializes the primary and secondary Maps used for initializing the keys.
     */

    public void initializeKeySetGroupKeys() throws com.ibm.websphere.crypto.KeyException
    {
        java.util.Map generationKeys = null;
        java.util.Map validationKeys = null;

        PublicKey tempPublicKey = null;
        PrivateKey tempPrivateKey = null;
        byte[] tempSharedKey = null;

        keySetGroupName = "ApplicationKeySetGroup";
        com.ibm.websphere.crypto.KeySetHelper ksh = com.ibm.websphere.crypto.KeySetHelper.getInstance();
        generationKeys = ksh.getLatestKeysForKeySetGroup(keySetGroupName);

        /***
        *  Latest keys: {
        *  KeyPair_3=com.ibm.websphere.crypto.KeyPair@64ec64ec,
        *  Secret_3=javax.crypto.spec.SecretKeySpec@fffe8aa7
        *  }
        ***/

        if (generationKeys != null)
        {
            Iterator iKeySet = generationKeys.keySet().iterator();

            while (iKeySet.hasNext())
            {
                String keyAlias = (String)iKeySet.next();

                Object key = generationKeys.get(keyAlias);

                if (key instanceof java.security.Key)
                {
                    tempSharedKey = ((java.security.Key)key).getEncoded();
                }
                else if (key instanceof com.ibm.websphere.crypto.KeyPair)
                {
                    java.security.Key publicKeyAsSecret =
    ((com.ibm.websphere.crypto.KeyPair)key).getPublicKey();
                    tempPublicKey = new PublicKey(publicKeyAsSecret.getEncoded());
                    java.security.Key privateKeyAsSecret =
    ((com.ibm.websphere.crypto.KeyPair)key).getPrivateKey();
                    tempPrivateKey = new PrivateKey(privateKeyAsSecret.getEncoded());
                }
```

```
            }

            // save these for use later, if necessary
            validationKeys = ksh.getAllKeysForKeySetGroup(keySetGroupName);

            /***
            *  All keys: {
            *  version_1=
            *      {Secret_1=javax.crypto.spec.SecretKeySpec@178cf,
            *       KeyPair_1=com.ibm.websphere.crypto.KeyPair@1c121c12},
            *  version_2=
            *      {Secret_2=javax.crypto.spec.SecretKeySpec@17a77,
            *       KeyPair_2=com.ibm.websphere.crypto.KeyPair@182e182e},
            *  version_3=
            *      {Secret_3=javax.crypto.spec.SecretKeySpec@fffe8aa7,
            *       KeyPair_3=com.ibm.websphere.crypto.KeyPair@4da04da0}
            *  }
            ***/
        }
        else
        {
            throw new com.ibm.websphere.crypto.KeyException("Could not generateKeys.");
        }
    }
```

Example: Developing a key or key pair generation class for automated key generation

A class that generates keys for cryptographic operations can be created automatically. With this capability, the key management infrastructure can maintain a list of keys for a predefined key set, and applications can access these keys.

You can schedule new key generation at predefined frequencies. Remember that key generation frequency affects the security of your data. For example, for persistent data, you might schedule key generation less frequently than for real time communications, which require that the keys be generated more often as old keys expire.

When you develop a key generation class, decide if you are creating a shared key or a key pair because this decision determines the interface you must use.

If you are developing shared keys, refer to the following example, which uses the KeyGenerator class to implement the com.ibm.websphere.crypto.KeyGenerator interface. The interface returns a java.security.Key key, which is stored as a SecretKey in a JCEKS keystore type. You can use any other keystore type that supports storing secret keys.

```
package com.ibm.test;

import java.util.*;
import com.ibm.ws.ssl.core.*;
import com.ibm.ws.ssl.config.*;
import com.ibm.websphere.crypto.KeyException;

public class KeyGenerator implements com.ibm.websphere.crypto.KeyGenerator
{
    private java.util.Properties customProperties = null;
    private java.security.Key secretKey = null;

    public KeyGenerator()
    {
    }

     /**
      * This method is called to pass any custom properties configured with
      * the KeySet to the implementation of this interface.
      *
      * @param java.util.Properties
      **/
    public void init (java.util.Properties customProps)
    {
        customProperties = customProps;
    }

     /**
      * This method is called whenever a key needs to be generated either
      * from the schedule or manually requested. The key is stored in the
      * KeyStore referenced by the configured KeySet that contains the
      * keyGenerationClass implementing this interface. The implementation of
      * this interface manages the key type. The user of the KeySet
      * must know the type that is returned by this keyGenerationClass.
```

```
     *
     * @return java.security.Key
     * @throws com.ibm.websphere.crypto.KeyException
     **/
    public java.security.Key generateKey () throws KeyException
    {
        try
        {
            // Assume generate3DESKey is there to create the key.
            byte[] tripleDESKey = generate3DESKey();
            secretKey = new javax.crypto.spec.SecretKeySpec(tripleDESKey, 0, 24, "3DES");

            if (secretKey != null)
            {
                return secretKey;
            }
            else
            {
                throw new com.ibm.websphere.crypto.KeyException ("Key could not be generated.");
            }
        }
        catch (Exception e)
        {
            e.printStackTrace();  // handle exception
        }
    }
}
```

If you are developing a key pair, refer to the following example, which uses the KeyPairGenerator class to implement the com.ibm.websphere.crypto.KeyPairGenerator interface.

```
package com.ibm.test;

import java.util.*;
import javax.crypto.spec.SecretKeySpec;
import com.ibm.websphere.crypto.KeyException;

/**
 * This implementation defines the method to generate a java.security.KeyPair.
 * When a keyGeneration class implements this method, the generateKeyPair method
 * is called and a KeyPair is stored in the keystore. The isKeyPair
 * attribute is ignored since the KeyGenerationClass is an
 * implementation of KeyPairGenerator. The isKeyPair attributes is for when
 * the keys already exist in a KeyStore, and are just read (not
 * generating them).
 *
 * @author IBM Corporation
 * @version WebSphere Application Server 6.1
 * @since WebSphere Application Server 6.1
 **/
public class KeyPairGenerator implements com.ibm.websphere.crypto.KeyPairGenerator
{
    private java.util.Properties customProperties = null;

    public KeyPairGenerator()
    {
    }

     /**
      * This method is called to pass any custom properties configured with
      * the KeySet to the implementation of this interface.
     *
      * @param java.util.Properties
      **/
    public void init (java.util.Properties customProps)
    {
        customProperties = customProps;
    }

     /**
     * This method is called whenever a key needs to be generated either
      * from the schedule or manually requested and isKeyPair=true in the KeySet
      * configuration. The key is stored in the KeyStore referenced by
      * the configured KeySet which contains the keyGenerationClass implementing
      * this interface. The implementation of this interface manages the
      * type of the key. The user of the KeySet must know the type that
      * is returned by this keyGenerationClass.
      *
      * @return com.ibm.websphere.crypto.KeyPair
      * @throws com.ibm.websphere.crypto.KeyException
      **/
    public com.ibm.websphere.crypto.KeyPair generateKeyPair () throws KeyException
    {
        try
        {
            java.security.KeyPair keyPair = generateKeyPair();

            // Store as SecretKeySpec
            if (keyPair != null)
```

```
            {
                java.security.PrivateKey privKey = keyPair.getPrivate();
                java.security.PublicKey pubKey = keyPair.getPublic();

                SecretKeySpec publicKeyAsSecretKey = new SecretKeySpec
                    (pubKey.getEncoded(), "RSA_PUBLIC");
                SecretKeySpec privateKeyAsSecretKey = new SecretKeySpec
                    (privKey.getEncoded(), "RSA_PRIVATE");

                com.ibm.websphere.crypto.KeyPair pair = new
    com.ibm.websphere.crypto.KeyPair(publicKeyAsSecretKey, privateKeyAsSecretKey);
                return pair;
            }
            else
            {
                throw new com.ibm.websphere.crypto.KeyException ("Key pair could
                not be generated.");
            }
        }
        catch (Exception e)
        {
            e.printStackTrace();  // handle exception
        }
    }
}
```

This interface returns a com.ibm.websphere.crypto.KeyPair key pair, which can contain either a X509Certificate and PrivateKey object or PublicKey and PrivateKey objects. If the com.ibm.websphere.crypto.KeyPair interface contains aX509Certificate and PrivateKey object, the certificate and private key are stored in the keystore. Consequently, they can use any KeyStore type.

If the com.ibm.websphere.crypto.KeyPair interface contains PublicKey and PrivateKey objects, you must convert the encoded values to the SecretKeySpec object in order to store them. The WebSphere Application Server runtime stores and retrieves the key pair as secret keys. The runtime converts the key pair back to PublicKey and PrivateKey objects when the server retrieves the pair during the handshake.

Use the following constructors to develop the com.ibm.websphere.crypto.KeyPair interface:

- Public and private constructor

```
public KeyPair(java.security.Key publicKey, java.security.Key privateKey)
```

- Certificate and private constructor.

```
public KeyPair(java.security.cert.Certificate[] certChain,
java.security.Key privateKey)
```

The previous example code shows the KeyPairGenerator class using the public and private constructor. Each call to this class generates a new and unique key pair, and this class is invoked by a KeySet to create a new key pair when `isKeyPair=true`. The version number in the key set increments each time it is called.

Key set groups collection

Use this page to manage groups of public, private, and shared keys. These key groups enable the application server to control multiple sets of Lightweight Third Party Authentication (LTPA) keys.

To view this administrative console page, click **Security > SSL certificate and key management**. Under Configuration settings, click **Manage endpoint security configurations > {Inbound | Outbound} > *ssl_configuration***. Under Related items, click **Key set groups**.

Table 111. Key set groups buttons. This table lists the key set groups buttons.

Button	Resulting action
New	Adds a key set group. A key set group combines one or more key sets together as a single key set group. It allows the generation of multiple different types of keys to occur at the same time. A single key set represents one type of key, so a key set group allows you to group the different types.
Delete	Deletes an existing key set group. You must be sure that there are no other references to this key set group before you delete it.

Table 111. Key set groups buttons (continued). This table lists the key set groups buttons.

Button	Resulting action
Generate keys	Generates keys for key set group. The system generates keys for each key set within the key set group so that the keys remain synchronized with each other in terms of version. You must configure a valid key generation class and a key store that is writable. See the com.ibm.websphere.crypto.KeySetHelper application programming interfaces (APIs) to enable the use of keys that are managed by a KeySetGroup or KeySet.

Key set group name
Specifies the name of the key set group used to reference it.

Automatically generate keys
Specifies that the keys are to be generated automatically on a schedule.

Key set groups settings
Use this page to create new key set groups.

To view this administrative console page, click **Security** > **SSL certificate and key management**. Under Configuration settings, click**Manage endpoint security configurations** > **{Inbound | Outbound}** > ***ssl_configuration***. Under Related items, click **Key set groups** > **New**.

Key set group name
Specifies the name of key set group used. This name can be referenced using the com.ibm.websphere.crypto.KeySetHelper API to retrieve the managed keys from an application.

Information	Value
Data type:	Text

Management scope
Specifies the scope where this Secure Sockets Layer (SSL) configuration is visible. For example, if you choose a specific node, then the configuration is only visible on that node and any servers that are part of that node.

Information	Value
Data type	List
Range:	Applicable scopes

Key sets
Specifies a set of key instances of the same type for use in cryptographic operations.

This setting has the following options:

Add Specifies to add the selected key set part of this key set group.

Remove
Specifies to remove the selection from the **Key sets** list.

Automatically generate keys
Specifies that the keys are generated automatically on a schedule. When a new key is generated, the `security.xml` is updated and saved by the runtime to track the key reference version. This can cause save conflicts when updating the same file from admin applications.

gotcha: Starting with Versions 6.1.0.23 and 7.0.0.3, the default value for this property is Disabled.

If you try to enable this property, and automatic synchronization is off in any node, the following administrative console message displays:

```
Warning: At least one node in the cell was unreachable or is not configured to automatically synchronize.
It is strongly recommended that you verify your node settings, and do not enable automatic
generation of LTPA keys while automatic synchronization is disabled on any node.
```

Information	Value
Default for Versions 7.0, and 7.0.0.1:	Enabled
Default for Versions 7.0.0.3 and higher:	Disabled

Scheduled time for generation

Specifies the scheduled time when the system generates selected key set group or groups. You can specify the scheduled time in hours and minutes; specify either A.M. or P.M., or specify 24-hour. You can also specify the day of the week you want the scheduled event to occur. It is recommended that you set this event to occur during a low peak time, especially for keys that are used by runtime for token validation.

Information	Value
Data type	Integer
Default:	8, 0 A.M.
Range:	1–12, with a A.M. or P.M. setting 0–59, with a 24-hour setting

Generate on a specific day

Specifies whether to have the generation occur on a specific day of the week. It is best to auto-generate keys during a low peak day.

This setting has the following options:

Weekday

Specifies the day of the week on which the expiration monitor will run if the Check on a specific day option is selected.

Information	Value
Default:	Sunday
Range:	Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday

Repeat interval

Specifies the period of time, in weeks, between each schedule time to check for expired certificates or the interval between schedule checks.

Information	Value
Default:	4 weeks

Generate at an interval

Specifies to generate keys at the specified frequency regardless of the day of the week on which generation occurs.

Information	Value
Default:	Disabled

This setting has the following options:

Repeat interval

Specifies the period of time, in days, between each schedule time to check for expired certificates or the interval between schedule checks.

Information	Value
Default:	7 days

Next start date
Specifies the date for the next scheduled check. This allows the deployment manager to be stopped and restarted without resetting the date.

Configuring the web server plug-in for Secure Sockets Layer

This topic documents the configuration that is necessary to instantiate a secure connection between the web server plug-in and the internal HTTP transport in the web container for the Application Server.

Before you begin

WebSphere Application Server has an internal HTTP transport that accepts HTTP requests. If you install an external HTTP server, the web server plug-in must forward requests from the external HTTP server to Application Server internal HTTP transport. Follow instructions provided by your HTTP vendor to install and configure your HTTP server. Test your HTTP server by accessing http://your-host-URL and https://your-host-URL. You should also have a web server plug-in installed. See the instructions for installing the HTTP Server and the web server plug-ins. They also describe how to enable the plug-in to load the correct libraries for Secure Socket layers (SSL) on Solaris x64.

Procedure

1. Create a directory on the web server host for storing the key ring file that is referenced by the plug-in and associated files, for example: *plugin_install_root*/etc/keys.
2. From the administrative console, click **Servers > Web servers**.
3. Select the web server name.
4. Click **Plug-in properties**.
5. Click **Manage keys and certificates** to access configuration options for your keys and certificates. By default, you can change your password used to protect the key store.
6. Click **OK**.
7. Click **Copy to web server keystore directory** to copy the keystore and to stash files to a managed web server. For non-managed web servers, use FTP to copy them.

   Note: You must copy the keystore file to the web server for the web server to function properly.
8. Optional: Under Additional Properties, you can also select one of the following:
   - **Signer certificates** - Use to add new certificates, delete certificates, extract certificates, and to retrieve certificates from a port.
   - **Personal certificates** - Use to create a new chained or self-signed certificate, delete a certificate, or to import and export a personal certificate.
   - **Personal certificate requests** - Use to manage personal certificate requests.
   - **Custom properties** - Use to define custom properties for the key store.

Results

The IBM HTTP Server plug-in and the internal Web server are configured for SSL.

Web server plug-in default configuration in SSL

When you create a new web server definition, WebSphere Application Server associates the web server plug-in with a Certificate Management Services (CMS) keystore for a specific node. The keystore contains all of the signers for the current cell with the self-signed or chained certificate, which belongs to the node.

The plug-in can communicate securely to WebSphere Application Server, even when the plug-in is configured with Secure Sockets Layer (SSL) client authentication enabled.

When you set the web server definition to webserver1 on node myhostNode01, WebSphere Application Server creates the keystore configuration. The keystore is scoped to the webserver1 server, which makes it visible to this server only. Other processes cannot use this keystore definition.

The following sample code from the `security.xml` file shows the configuration entries for the web server plug-in.

```
<keyStores xmi:id="KeyStore_1132357815719" name="CMSKeyStore"
password="{xor}HRYNFAtrbxEwOzpvbhw6MzM=" provider="IBMCMSProvider"
location="C:\WASX_e0540.11\AppServer\profiles\AppSrv01/config/cells
/myhostCell01/nodes/myhostNode01/servers/webserver1/plugin-key.kdb"
type="CMSKS" fileBased="true" createStashFileForCMS="true"
managementScope="ManagementScope_1132357815718"/>
<managementScopes xmi:id="ManagementScope_1132357815718" scopeName="
(cell):myhostCell01:(node):myhostNode01:(server):webserver1" scopeType="server"/>
```

The following sample code shows how the CMS keystore and stash file are generated in the `security.xml` file.

```
C:\WebSphere\AppServer\profiles\Dmgr01\config\cells\myhostCell01\nodes
\myhostNode01\servers\webserver1\plugin-key.kdb
C:\WebSphere\AppServer\profiles\Dmgr01\config\cells\myhostCell01
\nodes\myhostNode01\servers\webserver1\plugin-key.sth
```

The default password for the keystore is **WebAS**. You can change the default keystore password by using either the administrative console or the appropriate **AdminTask** command. The following sample code shows the **AdminTask** command that you can use to create this CMS keystore.

```
$AdminTask createCMSKeyStore /config/cells/myhostCell01/nodes/myhostNode01
/servers/webserver1/plugin-key.kdb myhost.austin.ibm.com
```

Note the following characteristics of the previous example:

- You can create only one CMSKeyStore entry for each management scope. If a CMS keystore already exists for scope (cell):myhostCell01:(node):myhostNode01:(server):webserver1, then you cannot create another CMSKeyStore entry
- The Uniform Resource Identifier (URI) for the keystore name is `/config/cells/myhostCell01/nodes/myhostNode01/servers/webserver1/plugin-key.kdb`
- The host name in the plug-in location is myhost.austin.ibm.com. WebSphere Application Server uses this name to create a chained certificate, if a chained certificate does not already exist for that particular node. If a chained certificate already exists for the node, then the certificate is put into the CMS keystore and all the signers from the cell are added, by default.

When additional nodes are federated, the signers for these nodes are not automatically added to each web server for the CMS keystore. For the web server plug-in to be able to communicate with a newly federated node, you must manually exchange signers with the CMSKeyStore keystore. Use the administrative console keystore certificate management function to exchange signers. For more information, see “Extracting a signer certificate from a personal certificate” on page 790.

Chapter 10. Developing extensions to the WebSphere security infrastructure

WebSphere Application Server provides various plug points so that you can extend the security infrastructure. Extending this security infrastructure involves several activities including: Developing custom user registries, developing applications that use programmatic security, and customizing web application login forms.

About this task

The following topics are covered in this section:

Procedure

- Developing custom user registries
- Developing applications that use programmatic security
- Customizing web application login forms
- Customizing application login forms with Java Authentication and Authorization Service (JAAS)
- Securing transports with Java Secure Sockets Extension (JSSE) and Java Cryptography Extension (JCE) programming interfaces
- Implementing tokens for security attribute propagation
- Implementing a custom authentication provider using JASPI

Developing stand-alone custom registries

This development provides considerable flexibility in adapting WebSphere Application Server security to various environments where some notion of a user registry, other than LDAP or Local OS, already exists in the operational environment.

Before you begin

WebSphere Application Server security supports the use of stand-alone custom registries in addition to the local operating system registry, stand-alone Lightweight Directory Access Protocol (LDAP) registries, and federated repositories for authentication and authorization purposes. A stand-alone custom-implemented registry uses the UserRegistry Java interface as provided by WebSphere Application Server. A stand-alone custom-implemented registry can support virtually any type or notion of an accounts repository from a relational database, flat file, and so on.

Implementing a stand-alone custom registry is a software development effort. Implement the methods that are defined in the com.ibm.websphere.security.UserRegistry interface to make calls to the appropriate registry to obtain user and group information. The interface defines a general set of methods for encapsulating a wide variety of registries. You can configure a stand-alone custom registry as the selected repository when configuring WebSphere Application Server security on the Global security panel.

In WebSphere Application Server Version 8.5, make sure that your implementation of the stand-alone custom registry does not depend on any WebSphere Application Server components such as data sources, Enterprise JavaBeans (EJB) and Java Naming and Directory Interface (JNDI). You can not have this dependency because security is initialized and enabled prior to most of the other WebSphere Application Server components during startup. If your previous implementation used these components, make a change that eliminates the dependency. For example, if your previous implementation used data sources to connect to a database, instead use the JDBC java.sql.DriverManager interface to connect to the database.

If your previous implementation uses data sources to connect to a database, change the implementation to use Java database connectivity (JDBC) connections.

Procedure

1. Implement all the methods in the interface except for the CreateCredential method, which is implemented by WebSphere Application Server.
2. Build your implementation.

   To compile your code, you need the *app_server_root*/plugins/com.ibm.ws.runtime.jar and the *app_server_root*/plugins/com.ibm.ws.security.crypto.jar files in your class path. For example:

   Windows

   ```
   app_server_root/java/bin/javac -classpath
   app_server_root/plugins/com.ibm.ws.runtime.jar;
   app_server_root/plugins/com.ibm.ws.security.crypto.jar your_implementation_file.java
   ```

   AIX HP-UX Linux Solaris

   ```
   app_server_root\java\bin\javac -classpath
   app_server_root\plugins\com.ibm.ws.runtime.jar:
   app_server_root\plugins\com.ibm.ws.security.crypto.jar your_implementation_file.java
   ```
3. Copy the class files that are generated in the previous step to the product class path.

   The preferred location is the following directory:

   - Windows *%install_root%*/lib/ext
   - AIX HP-UX Linux Solaris *%install_root%*\lib\ext

   directory. Copy these class files to all of the product process class paths.
4. To configure your implementation using the administrative console, follow the steps in topics about configuring stand-alone custom registries. This step is required to implement custom user registries.

Example

Viewing stand-alone custom registries.

Use these links to view registry examples.

A *stand-alone custom registry* is a customer-implemented registry that implements the UserRegistry Java interface, as provided by WebSphere Application Server. A custom-implemented registry can support virtually any type or form of an accounts repository from a relational database, flat file, and so on. The custom registry provides considerable flexibility in adapting WebSphere Application Server security to various environments where some form of a registry, other than a federated repository, Lightweight Directory Access Protocol (LDAP) registry, or local operating system registry, already exist in the operational environment.

What to do next

If you enable security, make sure that you complete the remaining steps:

1. Save and synchronize the configuration and restart all of the servers.
2. Try accessing some J2EE resources to verify that the custom registry implementation is correct.

Result.java file

This module is used by user registries in WebSphere Application Server when calling the getUsers and getGroups methods. The user registries use this method to set the list of users and groups and to indicate if more users and groups in the user registry exist than requested.

```
//
//  5639-D57, 5630-A36, 5630-A37, 5724-D18
// (C) COPYRIGHT International Business Machines Corp. 1997, 2005
//  All Rights Reserved * Licensed Materials - Property of IBM
```

```
//
package com.ibm.websphere.security;

import java.util.List;

public class Result implements java.io.Serializable {
    /**
      Default constructor
    */
    public Result() {
    }

    /**
       Returns the list of users and groups
       @return the list of users and groups
    */
    public List getList() {
      return list;
    }

    /**
       indicates if there are more users and groups in the registry
    */
    public boolean hasMore() {
      return more;
    }
    /**
       Set the flag to indicate that there are more users and groups
       in the registry to true
    */
    public void setHasMore() {
      more = true;
    }

    /*
      Set the list of users and groups
      @param list   list of users/groups
    */
    public void setList(List list) {
      this.list = list;
    }

    private boolean more = false;
    private List list;
}
```

UserRegistry.java files

The following file is a custom property that is used with a custom user registry.

For more information, see Configuring stand-alone custom registries.

```
// 5639-D57, 5630-A36, 5630-A37, 5724-D18
// (C) COPYRIGHT International Business Machines Corp. 1997, 2005
// All Rights Reserved * Licensed Materials - Property of IBM
//
// DESCRIPTION:
//
//    This file is the UserRegistry interface that custom registries in WebSphere
//    Application Server implement to enable WebSphere security to use the custom
//    registry.
//

package com.ibm.websphere.security;

import java.util.*;
import java.rmi.*;
import java.security.cert.X509Certificate;
import com.ibm.websphere.security.cred.WSCredential;

/**
 * Implementing this interface enables WebSphere Application Server Security
 * to use custom registries. This interface extends java.rmi.Remote because the
 * registry can be in a remote process.
 *
 * Implementation of this interface must provide implementations for:
*
* initialize(java.util.Properties)
* checkPassword(String,String)
* mapCertificate(X509Certificate[])
* getRealm
* getUsers(String,int)
* getUserDisplayName(String)
* getUniqueUserId(String)
* getUserSecurityName(String)
* isValidUser(String)
* getGroups(String,int)
* getGroupDisplayName(String)
```

```
* getUniqueGroupId(String)
* getUniqueGroupIds(String)
* getGroupSecurityName(String)
* isValidGroup(String)
* getGroupsForUser(String)
* getUsersForGroup(String,int)
* createCredential(String)
**/

public interface UserRegistry extends java.rmi.Remote
{

  /**
   * Initializes the registry. This method is called when creating the
   * registry.
   *
   * @param props the registry-specific properties with which to
   *              initialize the  custom registry
   * @exception CustomRegistryException
   *                    if there is any registry specific problem
   * @exception RemoteException
   *    as this extends java.rmi.Remote
   **/
   public void initialize(java.util.Properties props)
      throws CustomRegistryException,
             RemoteException;
  /**
   * Checks the password of the user. This method is called to authenticate a
   * user when the user's name and password are given.
   *
   * @param userSecurityName the name of the user
   * @param password the password of the user
   * @return a valid userSecurityName. Normally this is
   *   the name of same user whose password was checked but if the
   *  implementation wants to return any other valid
   *  userSecurityName in the registry it can do so
   * @exception CheckPasswordFailedException if userSecurityName/
   *  password combination does not exist in the registry
   * @exception CustomRegistryException if there is any registry specific
   *            problem
   * @exception RemoteException as this extends java.rmi.Remote
   **/
   public String checkPassword(String userSecurityName, String password)
      throws PasswordCheckFailedException,
             CustomRegistryException,
             RemoteException;

  /**
   * Maps a certificate (of X509 format) to a valid user in the registry.
   * This is used to map the name in the certificate supplied by a browser
   * to a valid userSecurityName in the registry
   *
   * @param cert the X509 certificate chain
   * @return the mapped name of the user userSecurityName
   * @exception CertificateMapNotSupportedException if the particular
   *            certificate is not supported.
   * @exception CertificateMapFailedException if the mapping of the
   *            certificate fails.
   * @exception CustomRegistryException if there is any registry specific
   *            problem
   * @exception RemoteException as this extends java.rmi.Remote
   **/
   public String mapCertificate(X509Certificate[] cert)
      throws CertificateMapNotSupportedException,
             CertificateMapFailedException,
             CustomRegistryException,
             RemoteException;

  /**
   * Returns the realm of the registry.
   *
   * @return the realm. The realm is a registry-specific string indicating
   *            the realm or domain for which this registry
   *            applies.  For example, for OS400 or AIX this would be the
   *            host name of the system whose user registry this object
   *            represents.
   *            If null is returned by this method realm defaults to the
   *            value of "customRealm". It is recommended that you use
   *            your own value for realm.
   * @exception CustomRegistryException if there is any registry specific
   *            problem
   * @exception RemoteException as this extends java.rmi.Remote
   **/
   public String getRealm()
      throws CustomRegistryException,
             RemoteException;

  /**
   * Gets a list of users that match a pattern in the registry.
   * The maximum number of users returned is defined by the limit
```

```
 * argument.
 * This method is called by administrative console and by scripting (command
 * line) to make available the users in the registry for adding them (users)
 * to roles.
 *
 * @parameter pattern the pattern to match. (For example., a* will match all
 *   userSecurityNames starting with a)
 * @parameter limit the maximum number of users that should be returned.
 *  This is very useful in situations where there are thousands of
 *            users in the registry and getting all of them at once is not
 *            practical. A value of 0 implies get all the users and hence
 *            must be used with care.
 * @return a Result object that contains the list of users
 *   requested and a flag to indicate if more users exist.
 * @exception CustomRegistryException if there is any registry specific
 *            problem
 * @exception RemoteException as this extends java.rmi.Remote
 **/
 public Result getUsers(String pattern, int limit)
    throws CustomRegistryException,
           RemoteException;

/**
 * Returns the display name for the user specified by userSecurityName.
 *
 * This method is called only when the user information displays
 * (information purposes only, for example, in the administrative console) and not used
 * in the actual authentication or authorization purposes. If there are no
 * display names in the registry return null or empty string.
 *
 * In WebSphere Application Server Version 4.0 custom registry, if you had a display
 * name for the user and  if it was different from the security name, the display name
 * was returned for the EJB methods getCallerPrincipal() and the servlet methods
 * getUserPrincipal() and  getRemoteUser().
 * In WebSphere Application Server Version 6.0 for the same methods the security
 * name is returned by default. This is the recommended way as the display name
 * is not unique and might create security holes.
 *
 * See the documentation for more information.
 *
 * @parameter userSecurityName the name of the user.
 * @return the display name for the user. The display name
 *   is a registry-specific string that represents a descriptive, not
 *  necessarily unique, name for a user. If a display name does
 *            not exist return null or empty string.
 * @exception EntryNotFoundException if userSecurityName does not exist.
 * @exception CustomRegistryException if there is any registry specific
 *            problem
 * @exception RemoteException as this extends java.rmi.Remote
 **/
 public String getUserDisplayName(String userSecurityName)
    throws EntryNotFoundException,
           CustomRegistryException,
           RemoteException;

/**
 * Returns the unique ID for a userSecurityName. This method is called when
 * creating a credential for a user.
 *
 * @parameter userSecurityName the name of the user.
 * @return the unique ID of the user. The unique ID for a user is
 *   the stringified form of some unique, registry-specific, data
 *  that serves to represent the user.  For example, for the UNIX
 *  user registry, the unique ID for a user can be the UID.
 * @exception EntryNotFoundException if userSecurityName does not exist.
 * @exception CustomRegistryException if there is any registry specific
 *            problem
 * @exception RemoteException as this extends java.rmi.Remote
 **/
 public String getUniqueUserId(String userSecurityName)
    throws EntryNotFoundException,
           CustomRegistryException,
           RemoteException;

/**
 * Returns the name for a user given its unique ID.
 *
 * @parameter uniqueUserId the unique ID of the user.
 * @return the userSecurityName of the user.
 * @exception EntryNotFoundException if the uniqueUserID does not exist.
 * @exception CustomRegistryException if there is any registry specific
 *            problem
 * @exception RemoteException as this extends java.rmi.Remote
 **/
 public String getUserSecurityName(String uniqueUserId)
    throws EntryNotFoundException,
           CustomRegistryException,
           RemoteException;

/**
```

```
 * Determines if the userSecurityName exists in the registry
 *
 * @parameter userSecurityName the name of the user
 * @return true if the user is valid. false otherwise
 * @exception CustomRegistryException if there is any registry specific
 *            problem
 * @exception RemoteException as this extends java.rmi.Remote
 **/
 public boolean isValidUser(String userSecurityName)
    throws CustomRegistryException,
           RemoteException;

/**
 * Gets a list of groups that match a pattern in the registry.
 * The maximum number of groups returned is defined by the limit
 * argument.
 * This method is called by the administrative console and scripting
 * (command line) to make available the groups in the registry for adding
 * them (groups) to roles.
 *
 * @parameter pattern the pattern to match. (For e.g., a* will match all
 *   groupSecurityNames starting with a)
 * @parameter limit the maximum number of groups to return.
 *  This is very useful in situations where there are thousands of
 *            groups in the registry and getting all of them at once is not
 *            practical. A value of 0 implies get all the groups and hence
 *            must be used with care.
 * @return a Result object that contains the list of groups
 *   requested and a flag to indicate if more groups exist.
 * @exception CustomRegistryException if there is any registry-specific
 *            problem
 * @exception RemoteException as this extends java.rmi.Remote
 **/
 public Result getGroups(String pattern, int limit)
    throws CustomRegistryException,
           RemoteException;

/**
 * Returns the display name for the group specified by groupSecurityName.
 *
 * This method may be called only when the group information displayed
 * (for example, the administrative console) and not used in the actual
 * authentication or authorization purposes. If there are no display names
 * in the registry return null or empty string.
 *
 * @parameter groupSecurityName the name of the group.
 * @return the display name for the group. The display name
 *   is a registry-specific string that represents a descriptive, not
 *  necessarily unique, name for a group. If a display name does
 *            not exist return null or empty string.
 * @exception EntryNotFoundException if groupSecurityName does not exist.
 * @exception CustomRegistryException if there is any registry specific
 *            problem
 * @exception RemoteException as this extends java.rmi.Remote
 **/
 public String getGroupDisplayName(String groupSecurityName)
    throws EntryNotFoundException,
           CustomRegistryException,
           RemoteException;

/**
 * Returns the unique ID for a group.

 * @parameter groupSecurityName the name of the group.
 * @return the unique ID of the group. The unique ID for
 *  a group is the stringified form of some unique,
 *  registry-specific, data that serves to represent the group.
 *  For example, for the UNIX user registry, the unique ID might
 *  be the GID.
 * @exception EntryNotFoundException if groupSecurityName does not exist.
 * @exception CustomRegistryException if there is any registry specific
 *            problem
 * @exception RemoteException as this extends java.rmi.Remote
 **/
 public String getUniqueGroupId(String groupSecurityName)
    throws EntryNotFoundException,
           CustomRegistryException,
           RemoteException;


/**
 * Returns the unique IDs for all the groups that contain the unique ID of
 * a user.
 * Called during creation of a user's credential.
 *
 * @parameter uniqueUserId the unique ID of the user.
 * @return a list of all the group unique IDs that the unique user ID
 *   belongs to. The unique ID for an entry is the stringified
 *  form of some unique, registry-specific, data that serves
 *  to represent the entry.  For example, for the
```

```
 *   UNIX user registry, the unique ID for a group could be the GID
 *  and the unique ID for the user could be the UID.
 * @exception EntryNotFoundException if unique user ID does not exist.
 * @exception CustomRegistryException if there is any registry specific
 *            problem
 * @exception RemoteException as this extends java.rmi.Remote
 **/
 public List getUniqueGroupIds(String uniqueUserId)
    throws EntryNotFoundException,
           CustomRegistryException,
           RemoteException;

/**
 * Returns the name for a group given its unique ID.
 *
 * @parameter uniqueGroupId the unique ID of the group.
 * @return the name of the group.
 * @exception EntryNotFoundException if the uniqueGroupId does not exist.
 * @exception CustomRegistryException if there is any registry-specific
 *            problem
 * @exception RemoteException as this extends java.rmi.Remote
 **/
 public String getGroupSecurityName(String uniqueGroupId)
    throws EntryNotFoundException,
           CustomRegistryException,
           RemoteException;

/**
 * Determines if the groupSecurityName exists in the registry
 *
 * @parameter groupSecurityName the name of the group
 * @return true if the groups exists, false otherwise
 * @exception CustomRegistryException if there is any registry specific
 *            problem
 * @exception RemoteException as this extends java.rmi.Remote
 **/
 public boolean isValidGroup(String groupSecurityName)
    throws CustomRegistryException,
           RemoteException;

/**
 * Returns the securityNames of all the groups that contain the user
 *
 * This method is called by administrative console and scripting
 * (command line) to verify the user entered for RunAsRole mapping belongs
 * to that role  in the roles to user mapping. Initially, the check is done
 * to see if the role contains the user. If the role does not contain the user
 * explicitly, this method is called to get the groups that this user
 * belongs to so that checks are made on the groups that the role contains.
 *
 * @parameter userSecurityName the name of the user
 * @return a List of all the group securityNames that the user
 *   belongs to.
 * @exception EntryNotFoundException if user does not exist.
 * @exception CustomRegistryException if there is any registry specific
 *            problem
 * @exception RemoteException as this extends java.rmi.Remote
 **/
 public List getGroupsForUser(String userSecurityName)
    throws EntryNotFoundException,
           CustomRegistryException,
           RemoteException;

/**
 * Gets a list of users in a group.
 *
 * The maximum number of users returned is defined by the limit
 * argument.
 *
 * This method is used by the WebSphere Business Integration
 * Server Foundation process choreographer when staff assignments
 * are modeled using groups.
 *

 * In rare situations where you are working with a user registry and it is not
 * practical to get all of the users from any of your groups (for example if
 * a large number of users exist) you can create the NotImplementedException
 * for those particular groups. Make sure that if the WebSphere Business
 * Integration Server Foundation Process Choreographer is installed (or
 * if installed later) that the users are not modeled using these particular groups.
 * If no concern exists about the staff assignments returning the users from
 * groups in the registry it is recommended that this method be implemented
 * without throwing the NotImplemented exception.
 *
 * @parameter groupSecurityName that represents the name of the group
 * @parameter limit the maximum number of users to return.
 *            This option is very useful in situations where lots of
 *            users are in the registry and getting all of them at
 *            once is not practical. A value of 0 means  get all of
 *            the users and must be used with care.
```

```
   * @return    a Result object that contains the list of users
   *            requested and a flag to indicate if more users exist.
   * @deprecated This method will be deprecated in the future.
   * @exception NotImplementedException create this exception in rare situations
   *            if it is not practical to get this information for any of the
   *            groups from the registry.
   * @exception EntryNotFoundException if the group does not exist in
   *            the registry
   * @exception CustomRegistryException if any registry-specific
   *            problem occurs
   * @exception RemoteException as this extends java.rmi.Remote interface
   **/
   public Result getUsersForGroup(String groupSecurityName, int limit)
      throws NotImplementedException,
             EntryNotFoundException,
             CustomRegistryException,
             RemoteException;

  /**
   * This method is implemented internally by the WebSphere Application Server
   * code in this release. This method is not called for the custom registry
   * implementations for this release. Return null in the implementation.
   *
   * Note that because this method is not called you can also return the
   * NotImplementedException as the previous documentation says.
   *
   **/
   public com.ibm.websphere.security.cred.WSCredential
                               createCredential(String userSecurityName)
      throws NotImplementedException,
      EntryNotFoundException,
             CustomRegistryException,
             RemoteException;
}
```

Implementing custom password encryption

WebSphere Application Server supports the use of custom password encryption.

Before you begin

An installation can implement any password encryption algorithm it chooses.

About this task

Complete the following steps to implement custom password encryption:

Procedure

1. Build your custom password encryption class. An example of a custom password encryption class follows.

```
// CustomPasswordEncryption
// Encryption and decryption functions
public interface CustomPasswordEncryption {
    public EncryptedInfo encrypt(byte[] clearText) throws PasswordEncryptException;
    public byte[] decrypt(EncryptedInfo cipherTextInfo) throws PasswordEncryptException;
    public void initialize(HashMap initParameters);
};
// Encapsulation of cipher text and label
public class EncryptedInfo {
    public EncryptedInfo(byte[] bytes, String keyAlias);
    public byte[] getEncryptedBytes();
    public String getKeyAlias();
};
```

2. If you need to custom encode passwords in property files, manually edit the PropFilePasswordEncoder.sh or PropFilePasswordEncoder.bat file.
   a. Use a file editor to open the PropFilePasswordEncoder.sh or PropFilePasswordEncoder.bat file.
   b. Locate the following lines near the end of the file:

```
"%JAVA_HOME%/bin/java" -Dcmd.properties.file=%TMPJAVAPROPFILE%
"-Dwas.install.root=%WAS_HOME%" com.ibm.ws.bootstrap.WSLauncher
com.ibm.ws.security.util.PropFilePasswordEncoder %1 %2
```

c. Add following lines to the call.

These custom properties will be passed to the command so that PropFilePasswordEncoder will look for custom encoding classes and utilize it.

```
-Dcom.ibm.wsspi.security.crypto.customPasswordEncryptionEnabled=true
-Dcom.ibm.wsspi.security.crypto.customPasswordEncryptionClass=(customEncoding class file)
```

The updated lines should look like the following lines:

```
"%JAVA_HOME%/bin/java" -Dcmd.properties.file=%TMPJAVAPROPFILE%
-Dcom.ibm.wsspi.security.crypto.customPasswordEncryptionEnabled=true
-Dcom.ibm.wsspi.security.crypto.customPasswordEncryptionClass=(customEncoding class file)
"-Dwas.install.root=%WAS_HOME%" com.ibm.ws.bootstrap.WSLauncher
com.ibm.ws.security.util.PropFilePasswordEncoder %1 %2
```

3. Enable custom password encryption.
   a. Set the custom property **com.ibm.wsspi.security.crypto.customPasswordEncryptionClass** to the name of the class that is to be given control.
   b. Enable the function. Set the custom property, **com.ibm.wsspi.security.crypto.customPasswordEncryptionEnabled** to `true`.

Results

Custom password encryption at the installation is complete.

Developing applications that use programmatic security

For some applications, declarative security is not sufficient to express the security model of the application. Use this topic to develop applications that use programmatic security.

About this task

IBM WebSphere Application Server provides security components that provide or collaborate with other services to provide authentication, authorization, delegation, and data protection. WebSphere Application Server also supports the security features that are described in the Java Platform, Enterprise Edition (Java EE) specification. An application goes through three stages before it is ready to run:

- Development
- Assembly
- Deployment

Most of the security for an application is configured during the assembly stage. The security that is configured during the assembly stage is called *declarative security* because the security is *declared* or *defined* in the deployment descriptors. The declarative security is enforced by the security runtime. For some applications, declarative security is not sufficient to express the security model of the application. For these applications, you can use *programmatic security*.

Procedure

1. Develop secure web applications. For more information, see “Developing with programmatic security APIs for web applications” on page 853.
2. Develop servlet filters for form login processing. For more information, see “Developing servlet filters for form login processing” on page 868.
3. Develop form login pages. For more information, see “Customizing web application login” on page 864.
4. Develop enterprise bean component applications. For more information, see “Developing with programmatic APIs for EJB applications” on page 860.
5. Develop with Java Authentication and Authorization Service to log in programmatically.

   For more information, see topics about developing programmatic logins with the Java Authentication and Authorization Service.
6. Develop your own Java EE security mapping module.

For more information, see topics about configuring programmatic logins for Java Authentication and Authorization Service.

7. Develop custom user registries. For more information, see “Developing stand-alone custom registries” on page 823.
8. Develop a custom interceptor for trust associations.

Protecting system resources and APIs (Java 2 security) for developing applications

Java 2 security is a programming model that is very pervasive and has a huge impact on application development.

Before you begin

Java 2 security is orthogonal to Java Platform, Enterprise Edition (Java EE) role-based security; you can disable or enable it independently of administrative security.

However, it does provide an extra level of access control protection on top of the Java EE role-based authorization. It particularly addresses the protection of system resources and application programming interfaces (API). Administrators need to consider the benefits against the risks of disabling Java 2 security.

The following recommendations are provided to help enable Java 2 security in a test or production environment:

1. Make sure the application is developed with the Java 2 security programming model. Developers have to know whether or not the APIs that are used in the applications are protected by Java 2 security. It is very important that the required permissions for the APIs used are declared in the policy file (`was.policy`), or the application fails to run when Java 2 security is enabled. Developers can reference the website for Development Kit APIs that are protected by Java 2 security. See the Programming model and decisions section of the Security: Resources for Learning topic to visit this website.
2. Make sure that migrated applications from previous releases are given the required permissions. Because Java 2 security is not supported or partially supported in previous WebSphere Application Server releases, applications developed prior to Version 5 most likely are not using the Java 2 security programming model. No easy way to find out all the required permissions for the application is available. The following are activities you can perform to determine the extra permissions that are required by an application:
   - Code review and code inspection
   - Application documentation review
   - Sandbox testing of migrated enterprise applications with Java 2 security enabled in a preproduction environment. Enable tracing in WebSphere Java 2 security manager to help determine the missing permissions in the application policy file. The trace specification is: `com.ibm.ws.security.core.SecurityManager=all=enabled`.
   - Use the com.ibm.websphere.java2secman.norethrow system property to aid debugging. Do not use this property in a production environment.

The default permission set for applications is the recommended permission set that is defined in the J2EE 1.3 Specification. The default is declared in the *app_server_root*`/profiles/`*`profile_name`*`/config/cells/`*`cell_name`*`/nodes/`*`node_name`*`/app.policy` policy file with permissions defined in the Development Kit (*`JAVA_HOME`*`/jre/lib/security/java.policy`) policy file that grant permissions to everyone. However, applications are denied permissions that are declared in the `profiles/`*`profile_name`*`/config/cells/`*`cell_name`*`/filter.policy` file. Permissions that are declared in the *filter.policy* file are filtered for applications during the permission check.

Define the required permissions for an application in a `was.policy` file and embed the `was.policy` file in the application enterprise archive (EAR) file as *YOURAPP.ear/META-INF/was.policy*, see “Configuring Java 2 security policy files” on page 834 for details.

The following steps describe how to enforce Java 2 security on the cell level for WebSphere Application Server, Network Deployment and the server level for WebSphere Application Server, Express

Procedure

1. Click **Security > Global security**. The Global security panel is displayed.
2. Select the **Use Java 2 security to restrict application access to local resources** option.
3. Click **OK** or **Apply**.
4. Click **Save** to save the changes.
5. Restart the server for the changes to take effect.

Results

Java 2 security is enabled and enforced for the servers. Java 2 security permission is selected when a Java 2 security protected API is called.

When to use Java 2 security

1. Enable protection on system resources, for example when opening or listening to a socket connection, reading or writing to operating system file systems, reading or writing Java virtual machine system properties, and so on.
2. Prevent application code from calling destructive APIs, for example, calling the System.exit method brings down the application server.
3. Prevent application code from obtaining privileged information (passwords) or gaining extra privileges (obtaining server credentials).

What to do next

The Java 2 security manager is enhanced to dump the Java 2 security permissions that are granted to all classes on the call stack when an application is denied access to a resource. The java.security.AccessControlException exception is created. However, this tracing capability is disabled by default. You can enable this capability by specifying the server trace service with the `com.ibm.ws.security.core.SecurityManager=all=enabled` trace specification. When the exception is created, the trace dump provides hints to determine whether the application is missing permissions or the product runtime code or the third-party libraries that are used are not properly marked as `privileged` when accessing Java 2 security-protected resources.

Using PolicyTool to edit policy files for Java 2 security

Use the **PolicyTool** utility to update policy files.

Before you begin

Java 2 security uses several policy files to determine the granted permission for each Java program. The Java Development Kit provides the **PolicyTool** tool to edit these policy files. This tool is recommended for editing any policy file to verify the syntax of its contents. Syntax errors in the policy file cause an AccessControlException exception when the application runs, including the server start. Identifying the cause of this exception is not easy because the user might not be familiar with the resource that has an access violation. Be careful when you edit these policy files.

Procedure

1. Start the **PolicyTool**.

   Windows For example, you can enter the following command at a Windows command prompt:

```
%{was.install.root}/java/jre/bin/policytool
```

   The **PolicyTool** window opens. The tool looks for the `java.policy` file in your home directory. If it does not exist, an error message displays.

   Click **OK**.

2. Click **File** > **Open**.
3. Navigate the directory tree in the **Open** window to pick up the policy file that you need to update. After selecting the policy file, click **Open**. The code base entries are listed in the window.
4. Create or modify the code base entry.
   a. Modify the existing code base entry by double-clicking the code base, or click the code base and click **Edit Policy Entry**. The Policy Entry window opens with the permission list defined for the selected code base.
   b. Create a new code base entry by clicking **Add Policy Entry**.

      The Policy Entry window opens. At the code base column, enter the code base information as a URL format.

      For example, you can enter:

```
app_server_root/InstalledApps/testcase.ear
```

      where the *app_server_root* variable represents your installation location.
5. Modify or add the permission specification.
   a. Modify the permission specification by double-clicking the entry that you want to modify, or by selecting the permission and clicking **Edit Permission**. The Permissions window opens with the selected permission information.
   b. Add a new permission by clicking **Add Permission**. The Permissions window opens. In the Permissions window are four rows for Permission, Target Name, Actions, and Signed By.
6. Select the permission from the Permission list. The selected permission displays. After a permission is selected, the Target Name, Actions, and Signed By fields automatically show the valid choices or they enable text input in the right text input area.
   a. Select **Target Name** from the list, or enter the target name in the right text input area.
   b. Select **Actions** from the list.
   c. Input **Signed By** if it is needed.

      Important: The Signed By keyword is not supported in the following policy files: `app.policy`, `spi.policy`, `library.policy`, `was.policy`, and `filter.policy` files. However, the Signed By keyword is supported in the following policy files: `#java.policy`, `server.policy`, and `client.policy` files. The Java Authentication and Authorization Service (JAAS) is not supported in the `app.policy`, `spi.policy`, `library.policy`, `was.policy`, and `filter.policy` files. However, the JAAS principal keyword is supported in a JAAS policy file when it is specified by the java.security.auth.policy Java virtual machine (JVM) system property.
7. Click **OK** to close the Permissions window. Modified permission entries of the specified code base display.
8. Click **Done** to close the window. Modified code base entries are listed. Repeat the previous steps until you complete editing.
9. Click **File** > **Save** after you finish editing the file.

Results

A policy file is updated. If any policy files need editing, use the **PolicyTool** utility. Do not edit the policy file manually. Syntax errors in the policy files can potentially cause application servers or enterprise applications to not start or function incorrectly. For the changes in the updated policy file to take effect, restart the Java processes.

Configuring Java 2 security policy files

Users can configure Java 2 security policy files so that the required permission is granted for the specified WebSphere Application Server enterprise application.

Before you begin

Java 2 security uses several policy files to determine the permissions for each Java programs.

See the Java 2 security policy files topic for the list of available policy files that are supported by WebSphere Application Server.

Two types of policy files are supported by WebSphere Application Server: dynamic policy files and static policy files. Static policy files provide the default permissions. Dynamic policy files provide application permissions. Six dynamic policy files are provided:

Table 112. Dynamic policy files. This table lists the dynamic policy files.

Policy file name	Description
app.policy	Contains default permissions for all of the enterprise applications in the cell. **Note:** Updates to the app.policy file only apply to the enterprise applications on the node to which the app.policy file belongs.
was.policy	Contains application-specific permissions for an WebSphere Application Server enterprise application. This file is packaged in an enterprise archive (EAR) file.
ra.xml	Contains connector application specific permissions for a WebSphere Application Server enterprise application. This file is packaged in a resource adapter archive (RAR) file.
spi.policy	Contains permissions for Service Provider Interface (SPI) or third-party resources that are embedded in WebSphere Application Server. The default contents grant everything. Update this file carefully when the cell requires more protection against SPI in the cell. This file is applied to all of the SPIs that are defined in the resources.xml file.
library.policy	Contains permissions for the shared library of enterprise applications.
filter.policy	Contains the list of permissions that require filtering from the was.policy file and the app.policy file in the cell. This filtering mechanism only applies to the was.policy and app.policy files.

In WebSphere Application Server, applications must have the appropriate thread permissions specified in the was.policy or app.policy file. Without the thread permissions specified, the application cannot manipulate threads and WebSphere Application Server creates a java.security.AccessControlException exception. The app.policy file applies to a specified node. If you change the permissions in one app.policy file, you must incorporate the new thread policy in the same file on the remaining nodes. Also, if you add the thread permissions to the app.policy file, you must restart WebSphere Application Server to enforce the new permissions. However, if you add the permissions to the was.policy file for a specific application, you do not need to restart WebSphere Application Server. An administrator must add the following code to a was.policy or app.policy file for an application to manipulate threads:

```
grant codeBase "file:${application}" {
  permission java.lang.RuntimePermission "stopThread";
  permission java.lang.RuntimePermission "modifyThread";
  permission java.lang.RuntimePermission "modifyThreadGroup";
};
```

Important: The Signed By keyword is not supported in the following policy files: app.policy, spi.policy, library.policy, was.policy, and filter.policy files. However, the Signed By keyword is supported in the following policy files:java.policy, server.policy, and client.policy files. The Java Authentication and Authorization Service (JAAS) is not supported in the app.policy, spi.policy, library.policy, was.policy, and filter.policy files. However, the JAAS principal keyword is supported in a JAAS policy file when it is specified by the java.security.auth.policy Java virtual machine (JVM) system property. You can statically set the authorization policy files in java.security.auth.policy with auth.policy.url.n=*URL*, where *URL* is the location of the authorization policy.

Procedure

1. Identify the policy file to update.
   - If the permission is required by an application, update the static policy file. Refer to “Configuring static policy files in Java 2 security” on page 847.
   - If the permission is required by all of the WebSphere Application Server enterprise applications in the node, refer to “spi.policy file permissions” on page 843.
   - If the permission is required only by specific WebSphere Application Server enterprise applications and the permission is required only by connector, update the `ra.xml` file. Refer to Refer to the Assembling resource adapter (connector) modules article for more information. Otherwise, update the `was.policy` file. Refer to “Configuring the was.policy file for Java 2 security” on page 840 and “Adding the was.policy file to applications for Java 2 security” on page 845.
   - If the permission is required by shared libraries, refer to “library.policy file permissions” on page 844.
   - If the permission is required by SPI libraries, refer to “spi.policy file permissions” on page 843.

   Tip: Pick up the policy file with the smallest scope. You can avoid giving an extra permission to the Java programs and protect the resources. You can update the `ra.xml` file or the `was.policy` file rather than the `app.policy` file. Use specific component symbols ($(ejbcomponent), ${webComponent},${connectorComponent} and ${jars}) than ${application} symbols. Update dynamic policy files, rather than static policy files.

   Add any permission that you never want granted to the WebSphere Application Server enterprise application in the cell to the `filter.policy` file. Refer to “filter.policy file permissions” on page 839.
2. Restart the WebSphere Application Server enterprise application.

Results

The required permission is granted for the specified WebSphere Application Server enterprise application.

Example

If an WebSphere Application Server enterprise application in a cell requires permissions, some of the dynamic policy files need updating. The symptom of the missing permission is the java.security.AccessControlException exception. The missing permission is listed in the exception data, which will appear as one line, but is split in sections below for readability.

When a Java program receives this exception and adding this permission is justified, add a permission to an adequate dynamic policy file.

The previous permission information lines are split for the illustration. Enter the permission on one line.

To decide whether to add a permission, refer to the Access control exception for Java 2 security topic.

app.policy file permissions:

Java 2 security uses several policy files to determine the granted permissions for each Java program. The union of the permissions that are contained in these following files is applied to the WebSphere Application Server enterprise application. This union determines the granted permissions.

For the list of available policy files that are supported by WebSphere Application Server, see the topic about Java 2 security policy files. The `app.policy` file is a default policy file that is shared by all of the WebSphere Application Server enterprise applications. The union of the permissions that are contained in the following files is applied to the WebSphere Application Server enterprise application:

- Any policy file that is specified in the policy.url.* properties in the `java.security` file.
- The `app.policy` files, which are managed by configuration and file replication services.

- The server.policy file.
- The java.policy file.
- The application was.policy file.
- The permission specification of the ra.xml file.
- The shared library, which is the library.policy file.

In WebSphere Application Server, applications that manipulate threads must have the appropriate thread permissions specified in the was.policy or app.policy file. Without the thread permissions specified, the application cannot manipulate threads and WebSphere Application Server creates a java.security.AccessControlException exception. If an administrator adds thread permissions to the app.policy file, the permission change requires a restart of the WebSphere Application Server. An administrator must add the following code to a was.policy or app.policy file for an application to manipulate threads:

```
grant codeBase "file:${application}" {
  permission java.lang.RuntimePermission "stopThread";
  permission java.lang.RuntimePermission "modifyThread";
  permission java.lang.RuntimePermission "modifyThreadGroup";
};
```

Important: The Signed By and the Java Authentication and Authorization Service (JAAS) principal keywords are not supported in the app.policy file. However, the Signed By keyword is supported in the following files: java.policy, server.policy, and the client.policy files. The JAAS principal keyword is supported in a JAAS policy file when it is specified by the java.security.auth.policy Java virtual machine (JVM) system property. You can statically set the authorization policy files in the java.security.auth.policy property with auth.policy.url.n=URL where URL is the location of the authorization policy.

If the default permissions for enterprise applications (the union of the permissions that is defined in the java.policy file, the server.policy file and the app.policy file) are enough; no action is required. The default app.policy file is used automatically. If a specific change is required to all of the enterprise applications in the cell, update the app.policy file. Syntax errors in the policy files cause start failures in the application servers. Edit these policy files carefully.

Note: Updates to the app.policy file only apply to the enterprise applications on the node to which the app.policy file belongs.

To extract the policy file, use a command prompt to enter the following command on one line using the appropriate variable values for your environment:

Windows

```
wsadmin> set obj [$AdminConfig extract cells/cell_name/node/node_name/app.policy c:\temp\test\app.policy]
```

Edit the extracted app.policy file with the Policy Tool. For more information, see “Using PolicyTool to edit policy files for Java 2 security” on page 833. Changes to the app.policy file are local for the node.

To check in the policy file, use a command prompt to enter the following command on one line using the appropriate variable values for your environment:

Windows

```
wsadmin> $AdminConfig checkin cells/cell_name/nodes/node_name/app.policy c:\temp\test\app.policy $obj
```

Table 113. Symbols used to associate permission lists to a specific type of resource. Several product-reserved symbols are defined to associate the permission lists to a specific type of resource.

Symbol	Meaning
file:${application}	Permissions apply to all resources within the application
file:${jars}	Permissions apply to all utility Java archive (JAR) files within the application

Table 113. Symbols used to associate permission lists to a specific type of resource (continued). Several product-reserved symbols are defined to associate the permission lists to a specific type of resource.

Symbol	Meaning
file:${ejbComponent}	Permissions apply to enterprise bean resources within the application
file:${webComponent}	Permissions apply to web resources within the application
file:${connectorComponent}	Permissions apply to connector resources both within the application and within stand-alone connector resources.

Table 114. Symbols provided to specify the path and name for the java.io.FilePermission permission. Five embedded symbols are provided to specify the path and name for the java.io.FilePermission permission. These symbols enable flexible permission specifications. The absolute file path is fixed after the installation of the application.

Symbol	Meaning
${app.installed.path}	Path where the application is installed
${was.module.path}	Path where the module is installed
${current.cell.name}	Current cell name
${current.node.name}	Current node name
${current.server.name}	Current server name

Tip: You cannot use the `${was.module.path}` in the `${application}` entry.

Attention: In the following code sample, the first two lines that are related to java.io.FilePermission permission are split into two lines for illustrative purposes only.

```
grant codeBase "file:${application}" {
  // The following are required by JavaMail
  permission java.io.FilePermission "${was.install.root}${/}lib${/}activation-impl.jar", "read";
  permission java.io.FilePermission "${was.install.root}${/}lib${/}mail-impl.jar", "read";
};

grant codeBase "file:${jars}" {
  permission java.net.SocketPermission "*", "connect";
  permission java.util.PropertyPermission "*", "read";
};

grant codeBase "file:${connectorComponent}" {
  permission java.net.SocketPermission "*", "connect";
  permission java.util.PropertyPermission "*", "read";
};

grant codeBase "file:${webComponent}" {
  permission java.io.FilePermission "${was.module.path}${/}-", "read, write";
  permission java.lang.RuntimePermission "loadLibrary.*";
  permission java.lang.RuntimePermission "queuePrintJob";
  permission java.net.SocketPermission "*", "connect";
  permission java.util.PropertyPermission "*", "read";
};

grant codeBase "file:${ejbComponent}" {
  permission java.lang.RuntimePermission "queuePrintJob";
  permission java.net.SocketPermission "*", "connect";
  permission java.util.PropertyPermission "*", "read";
};
```

If all of the WebSphere Application Server enterprise applications in a cell require permissions that are not defined as defaults in the `java.policy` file, the `server.policy` file and the `app.policy` file, then update the `app.policy` file. The symptom of a missing permission is the `java.security.AccessControlException` exception.

Note: Updates to the app.policy file only apply to the enterprise applications on the node to which the app.policy file belongs.

Windows The missing permission is listed in the exception data, for example,

```
java.security.AccessControlException: access denied
(java.io.FilePermission
C:\WebSphere\AppServer\java\jre\lib\ext\mail.jar read)
```

When a Java program receives this exception and adding this permission is justified, add a permission to the `server.policy` file, for example:

Windows

```
grant codeBase "file:user_client_installed_location" {
  permission java.io.FilePermission
"C:\WebSphere\AppServer\java\jre\lib\ext\mail.jar", "read";
 };
```

The previous permission information lines are split for the illustration. You actually enter the permission on one line.

To decide whether to add a permission, refer to the AccessControlException topic.

Restart all WebSphere Application Server enterprise applications to ensure that the updated `app.policy` file takes effect.

filter.policy file permissions:

Java 2 security uses several policy files to determine the granted permission for each Java program. Java 2 security policy filtering is only in effect when Java 2 security is enabled.

Before modifying the `filter.policy` file, you must start the wsadmin tool.

Refer to “Protecting system resources and APIs (Java 2 security) for developing applications” on page 832. The filtering policy defined in the `filter.policy` file is cell wide. The `filter.policy` file is the only policy file that is used when restricting the permission instead of granting permission. The permissions that are listed in the filter policy file are filtered out from the `app.policy` file and the `was.policy` file. Permissions that are defined in the other policy files are not affected by the `filter.policy` file.

When a permission is filtered out, an audit message is logged. However, if the permissions that are defined in the `app.policy` file and the `was.policy` file are compound permissions like the java.security.AllPermission permission, for example, the permission is not removed. A warning message is logged. If the Issue Permission Warning flag is enabled (default) and if the `app.policy` file and the `was.policy` file contain custom permissions (non-Java API permission, the permission package name begins with characters other than `java` or `javax`), a warning message is logged and the permission is not removed. You can change the value of the `Warn if applications are granted custom permissions` option on the Global security panel. It is not recommended that you use the `AllPermission` permission for the enterprise application.

Some default permissions that are defined in the `filter.policy` file. These permissions are the minimal ones that are recommended by the product. If more permissions are added to the `filter.policy` file, certain operations can fail for enterprise applications. Add permissions to the `filter.policy` file carefully.

You cannot use the Policy Tool to edit the `filter.policy` file. Editing must be completed in a text editor. Be careful and verify that no syntax errors exist in the `filter.policy` file. If any syntax errors exist in the `filter.policy` file, the file is not loaded by the product security runtime, which implies that filtering is disabled.

To extract the `filter.policy` file, enter the following command using information from your environment:

set obj [$AdminConfig extract cells/cell_name/filter.policy c:/temp/test/filter.policy]

To check in the policy file, enter the following command using information from your environment:

$AdminConfig checkin cells/cell_name/filter.policy c:/temp/test/filter.policy $obj

An updated `filter.policy` file is applied to all of the WebSphere Application Server enterprise applications after the servers are restarted. The `filter.policy` file is managed by configuration and file replication services.

The `filter.policy` file that is supplied by WebSphere Application Server resides at: *app_server_root*/`profiles/profile_name/config/cells/cell_name/filter.policy`.

This fill contains these permissions as defaults:

```
filterMask {
permission java.lang.RuntimePermission "exitVM";
permission java.lang.RuntimePermission "setSecurityManager";
permission java.security.SecurityPermission "setPolicy";
permission javax.security.auth.AuthPermission "setLoginConfiguration"; };
runtimeFilterMask {
permission java.lang.RuntimePermission "exitVM";
permission java.lang.RuntimePermission "setSecurityManager";
permission java.security.SecurityPermission "setPolicy";
permission javax.security.auth.AuthPermission "setLoginConfiguration"; };
```

The permissions that are defined in `filterMask` filter are for static policy filtering. The security runtime tries to remove the permissions from applications during application startup. Compound permissions are not removed, but are issued with a warning, and application deployment is stopped if applications contain permissions that are defined in the `filterMask` filter, and if scripting is used. The runtimeFilterMask filter defines permissions that are used by the security runtime to deny access to those permissions to application thread. Do not add more permissions to the runtimeFilterMask filter. Application start failure or incorrect functioning might result. Be careful when adding more permissions to the runtimeFilterMask filter. Usually, you only need to add permissions to the `filterMask` stanza.

WebSphere Application Server relies on the filter policy file to restrict or disallow certain permissions that can compromise the integrity of the system. For instance, WebSphere Application Server considers the exitVM and setSecurityManager permissions as those permissions that most applications never have. If these permissions are granted, the following scenarios are possible:

exitVM
: A servlet, JavaServer Pages (JSP) file, enterprise bean, or other library that is used by the aforementioned might call the System.exit API and cause the entire WebSphere Application Server process to terminate.

setSecurityManager
: An application might install its own security manager and either grant more permissions or bypass the default policy that the WebSphere Application Server security manager enforces.

Important: In application code, do not use the setSecurityManager permission to set a security manager. When an application uses the setSecurityManager permission, a conflict exists with the internal security manager within WebSphere Application Server. If you must set a security manager in an application for Remote Method Invocation (RMI) purposes, you also must select the Use Java 2 security to restrict application access to local resources option on the Global security panel within the WebSphere Application Server administrative console. WebSphere Application Server then registers a security manager, which the application code can verify is registered by using the System.getSecurityManager application programming interface (API).

For the updated `filter.policy` file to take effect, restart related Java processes.

Configuring the was.policy file for Java 2 security:

You should update the was.policy file if the application has specific resources to access.

Before you begin

Java 2 security uses several policy files to determine the granted permission for each Java program. The was.policy file is an application-specific policy file for WebSphere Application Server enterprise applications. This file is embedded in the META-INF/was.policy enterprise archive (.EAR) file. The was.policy file is located in:

profile_root/config/cells/*cell_name*/applications/
ear_file_name/deployments/*application_name*/META-INF/was.policy

See Java 2 security policy files for the list of available policy files that are supported by WebSphere Application Server Version 6.1.

The union of the permissions that are contained in the following files is applied to the WebSphere Application Server enterprise application:

- Any policy file that is specified in the policy.url.* properties in the java.security file.
- The app.policy files, which are managed by configuration and file replication services.
- The server.policy file.
- The java.policy file.
- The application was.policy file.
- The permission specification of the ra.xml file.
- The shared library, which is the library.policy file.

Table 115. Symbols defined to associate permission lists to a specific type of resource. Several product-reserved symbols are defined to associate the permission lists to a specific type of resource.

Symbol	Definition
file:${application}	Permissions apply to all resources used within the application.
file:${jars}	Permissions apply to all utility Java archive (JAR) files within the application
file:${ejbComponent}	Permissions apply to enterprise bean resources within the application
file:${webComponent}	Permissions apply to web resources within the application
file:${connectorComponent}	Permissions apply to connector resources within the application

In WebSphere Application Server, applications that manipulate threads must have the appropriate thread permissions specified in the was.policy or app.policy file. Without the thread permissions specified, the application cannot manipulate threads and WebSphere Application Server creates a java.security.AccessControlException exception. If you add the permissions to the was.policy file for a specific application, you do not need to restart WebSphere Application Server. An administrator must add the following code to a was.policy or app.policy file for an application to manipulate threads:

```
grant codeBase "file:${application}" {
  permission java.lang.RuntimePermission "stopThread";
  permission java.lang.RuntimePermission "modifyThread";
  permission java.lang.RuntimePermission "modifyThreadGroup";
};
```

An administrator can add the thread permissions to the app.policy file, but the permission change requires a restart of WebSphere Application Server.

Important: The Signed By and the Java Authentication and Authorization Service (JAAS) principal keywords are not supported in the was.policy file. The Signed By keyword is supported in the java.policy, server.policy, and client.policy policy file. The JAAS principal keyword is supported in a JAAS policy file when it is specified by the java.security.auth.policy Java virtual machine (JVM) system property. You can statically set the authorization policy files in the java.security.auth.policy file with the auth.policy.url.n=*URL*, where *URL* is the location of the authorization policy.

Other than these blocks, you can specify the module name for granular settings. For example,

```
grant codeBase "file:DefaultWebApplication.war" {
  permission java.security.SecurityPermission "printIdentity";
};

grant codeBase "file:IncCMP11.jar" {
  permission java.io.FilePermission
      "${user.install.root}${/}bin${/}DefaultDB${/}-",
      "read,write,delete";
};
```

Table 116. Embedded symbols provided to specify the path and name for the java.io.FilePermission permission. Five embedded symbols are provided to specify the path and name for the java.io.FilePermission permission. These symbols enable flexible permission specification. The absolute file path is fixed after the application is installed.

Symbol	Definition
${app.installed.path}	Path where the application is installed
${was.module.path}	Path where the module is installed
${current.cell.name}	Current cell name
${current.node.name}	Current node name
${current.server.name}	Current server name

About this task

If the default permissions for the enterprise application are enough, an action is not required. The default permissions are a union of the permissions that are defined in the `java.policy` file, the `server.policy` file, and the `app.policy` file. If an application has specific resources to access, update the `was.policy` file. The first two steps assume that you are creating a new policy file.

Tip: Syntax errors in the policy files cause the application server to fail. Use care when editing these policy files.

Procedure

1. Create or edit a new `was.policy` file by using the PolicyTool. For more information, see “Using PolicyTool to edit policy files for Java 2 security” on page 833.
2. Package the `was.policy` file into the enterprise archive (EAR) file.

   For more information, see “Adding the was.policy file to applications for Java 2 security” on page 845.The following instructions describe how to import a `was.policy` file.

   a. Import the EAR file into an assembly tool.
   b. Open the Project Navigator view.
   c. Expand the EAR file and click **META-INF**. You might find a `was.policy` file in the META-INF directory. If you want to delete the file, right-click the file name and select **Delete**.
   d. At the bottom of the Project Navigator view, click **J2EE Hierarchy**.
   e. Import the `was.policy` file by right-clicking the **Modules** directory within the deployment descriptor and by clicking **Import > Import > File system**.
   f. Click **Next**.
   g. Enter the path name to the `was.policy` file in the **From directory** field or click **Browse** to locate the file.
   h. Verify that the path directory that is listed in the **Into directory** field lists the correct `META-INF` directory.
   i. Click **Finish**.
   j. To validate the EAR file, right-click the EAR file, which contains the Modules directory, and click **Run Validation**.
   k. To save the new EAR file, right-click the EAR file, and click **Export > Export EAR file**. If you do not save the revised EAR file, the EAR file will contain the new `was.policy` file. However, if the workspace becomes corrupted, you might lose the revised EAR file.

l. To generate deployment code, right-click the EAR file and click **Generate Deployment Code**.

3. Update an existing installed application, if one already exists. Modify the `was.policy` file with the Policy Tool. For more information, see “Using PolicyTool to edit policy files for Java 2 security” on page 833.

Results

The updated `was.policy` file is applied to the application after the application restarts.

Example

When a Java program receives this exception and adding this permission is justified, add the following permission to the `was.policy` file:

To determine whether to add a permission, see Access control exception for Java 2 security.

What to do next

Restart all applications for the updated `app.policy` file to take effect.

spi.policy file permissions:

Java 2 security uses several policy files to determine the granted permission for each Java program.

For the list of available policy files that are supported by WebSphere Application Server Version 6.0.x, see Java 2 security policy files.

Because the default permission for the Service Provider Interface (SPI) is the AllPermission permission, the only reason to update the `spi.policy` file is a restricted SPI permission. When a change in the `spi.policy` is required, complete the following steps.

Syntax errors in the policy files cause the application server to fail. Edit these policy files carefully.

Important: Do not place the codebase keyword or any other keyword after the filterMask and runtimeFilterMask keywords. The Signed By and the Java Authentication and Authorization Service (JAAS) Principal keywords are not supported in the `spi.policy` file. The Signed By keyword is supported in the `java.policy`, `server.policy`, and `client.policy` policy files. The JAAS Principal keyword is supported in a JAAS policy file that is specified by the java.security.auth.policy Java virtual machine (JVM) system property. You can statically set the authorization policy files in `java.security.auth.policy` with auth.policy.url.n=URL, where URL is the location of the authorization policy.

To extract the `filter.policy` file, enter the following command using information from your environment:

set obj [$AdminConfig extract `profiles/profile_name/cells/cell_name/nodes/node_name/spi.policy` c:/temp/test/spi.policy]

Edit the file using the Policy Tool. For more information, see “Using PolicyTool to edit policy files for Java 2 security” on page 833.

To check in the policy file, enter the following command using information from your environment:

The updated `spi.policy` is applied to the Service Provider Interface (SPI) libraries after the Java process is restarted.

$AdminConfig checkin `profiles/profile_name/cells/cell_name/nodes/node_name/spi.policy` c:/temp/test/spi.policy $obj

Examples

The spi.policy file is the template for SPIs or third-party resources embedded in the product. Examples of SPIs are Java Message Services (JMS) (MQSeries®) and Java database connectivity (JDBC) drivers. They are specified in the resources.xml file. The dynamic policy grants the permissions that are defined in the spi.policy file to the class paths defined in the resources.xml file. The union of the permission that is contained in the java.policy file and the spi.policy file are applied to the SPI libraries. The spi.policy files are managed by configuration and file replication services.

You can find the spi.policy file that is supplied by WebSphere Application Server in the following location: *app_server_root*/profiles/profile_name/config/cells/cell_name/nodes/node_name/spi.policy. This file contains the following default permission:

```
grant {
  permission java.security.AllPermission;
};
```

Restart the related Java processes for the changes in the spi.policy file to become effective.

library.policy file permissions:

Java 2 security uses several policy files to determine the granted permission for each Java program.

For the list of available policy files that are supported by WebSphere Application Server, see Java 2 security policy files.

The library.policy file is the template for shared libraries (Java library classes). Multiple enterprise applications can define and use shared libraries. Refer to Managing shared libraries for information on how to define and manage the shared libraries.

If the default permissions for a shared library (union of the permissions defined in the java.policy file, the app.policy file and the library.policy file) are enough, no action is required. The default library policy is picked up automatically. If a specific change is required to share a library in the cell, update the library.policy file.

Syntax errors in the policy files cause the application server to fail. Edit these policy files carefully.

Important: Do not place the codebase keyword or any other keyword after the grant keyword. The Signed By keyword and the Java Authentication and Authorization Service (JAAS) Principal keyword are not supported in the library.policy file. The Signed By keyword is supported in the java.policy, the server.policy, and the client.policy policy files. The JAAS Principal keyword is supported in a JAAS policy file when it is specified by the Java virtual machine (JVM) system property, java.security.auth.policy. You can statically set the authorization policy files in the java.security.auth.policy file with auth.policy.url.n=URL where URL is the location of the authorization policy.

To extract the policy file, use a command prompt to enter the following command using the appropriate variable values for your environment:The previous two lines were split onto two lines for illustrative purposes only.

wsadmin> set obj [$AdminConfig extract cells/cell_name/nodes/
node_name/library.policy c:/temp/test/library.policy]

Edit the extracted library.policy file with the Policy Tool. For more information, see “Using PolicyTool to edit policy files for Java 2 security” on page 833.

To check in the policy file, use a command prompt to enter the following command using the appropriate variable values for your environment:An updated `library.policy` is applied to shared libraries after the servers restart.

```
wsadmin> $AdminConfig checkin cells/cell_name/nodes/node_name/library.policy
                    c:/temp/test/library.policy $obj
```

Example

The union of the permission that is contained in the `java.policy` file, the `app.policy` file, and the `library.policy` file are applied to the shared libraries. The `library.policy` file is managed by configuration and file replication services.

The `library.policy` file are supplied by WebSphere Application Server resides at: *app_server_root*/`config/cells/cell_name/nodes/node_name/` directory. The file contains an empty permission entry as a default. For example:

```
grant {
  };
```

If the shared library in a cell requires permissions that are not defined as defaults in the`java.policy` file, the `app.policy` file and the `library.policy` file, update the `library.policy` file. The missing permission causes the java.security.AccessControlException exception. The missing permission is listed in the exception data.

Windows For example:

```
java.security.AccessControlException: access denied (java.io.FilePermission
app_server_root/lib/mail-impl.jar read)
```

The previous lines are split into two lines for illustrative purposes only. The *app_server_root* variable represents your installation directory.

When a Java program receives this exception and adding this permission is justified, add a permission to the `library.policy` file.

Windows For example:

```
grant { permission java.io.FilePermission "app_server_root/lib/mail-impl.jar", "read"; };
```

The previous lines are split into two lines for illustrative purposes only. The *app_server_root* variable represents your installation directory.

To decide whether to add a permission, refer to Access control exception for Java 2 security.

Restart the related Java processes for the changes in the `library.policy` file to become effective.

Adding the was.policy file to applications for Java 2 security:

An application might need a `was.policy` file if it accesses resources that require more permissions than those granted in the default `app.policy` file.

About this task

When Java 2 security is enabled for a WebSphere Application Server, all the applications that run on WebSphere Application Server undergo a security check before accessing system resources. An application might need a `was.policy` file if it accesses resources that require more permissions than those granted in the default `app.policy` file. By default, the product security reads an `app.policy` file that is

located in each node and grants the permissions in the app.policy file to all the applications. Include any additional required permissions in the was.policy file. The was.policy file is only required if an application requires additional permissions.

The default policy file for all applications is specified in the app.policy file. This file is provided by the product security, is common to all applications, and you do not change this file. Add any new permissions that are required for an application in the was.policy file.

The app.policy file supplied by WebSphere Application Server resides at *app_server_root*/config/cells/profile/profile_name/config/*cell_name*/nodes/*node_name*/app.policy. The contents of the app.policy file are presented in the following example:

Attention: In the following code sample, the two permissions that are required by JavaMail are split onto two lines for illustration only. You actually enter the permission on one line.

```
// The following permissions apply to all the components under the application.

grant codeBase "file:${application}" {
   // The following are required by JavaMail

  permission java.io.FilePermission "
        ${was.install.root}${/}lib${/}activation-impl.jar",
"read";

  permission java.io.FilePermission "
        ${was.install.root}${/}lib${/}mail-impl.jar","read";

  };
   // The following permissions apply to all utility .jar files  (other
   //  than enterprise beans JAR files) in the application.
grant codeBase "file:${jars}" {
  permission java.net.SocketPermission "*", "connect";
  permission java.util.PropertyPermission "*", "read";
};

// The following permissions apply to connector resources within the application
grant codeBase "file:${connectorComponent}" {
  permission java.net.SocketPermission "*", "connect";
  permission java.util.PropertyPermission "*", "read";
};

// The following permissions apply to all the web modules (.war files)
// within the application.
grant codeBase "file:${webComponent}" {
  permission java.io.FilePermission "${was.module.path}${/}-", "read, write";
       //  where "was.module.path" is the path where the web module is
       //  installed. Refer to Dynamic policy concepts for other symbols.
  permission java.lang.RuntimePermission "loadLibrary.*";
  permission java.lang.RuntimePermission "queuePrintJob";
  permission java.net.SocketPermission "*", "connect";
  permission java.util.PropertyPermission "*", "read";
};

// The following permissions apply to all the EJB modules within the application.
grant codeBase "file:${ejbComponent}" {
 permission java.lang.RuntimePermission "queuePrintJob";
 permission java.net.SocketPermission "*", "connect";
 permission java.util.PropertyPermission "*", "read";
};
```

If additional permissions are required for an application or for one or more modules of an application, use the was.policy file for that application. For example, use codeBase of ${application} and add required permissions to grant additional permissions to the entire application. Similarly, use codeBase of ${webComponent} and ${ejbComponent} to grant additional permissions to all the web modules and all the enterprise bean modules in the application. You can assign additional permissions to each module (.war file or .jar file), as shown in the following example.

This example illustrates adding extra permissions for an application in the was.policy file:

Attention: In the following code sample, the permission for the EJB module was split onto two lines for illustration only. You actually enter the permission on one line.

```
// grant additional permissions to a web module
grant codeBase " file:aWebModule.war" {
 permission java.security.SecurityPermission "printIdentity";
```

```
};

// grant additional permission to an EJB module
grant codeBase "file:aEJBModule.jar"  {
    permission java.io.FilePermission "
       ${user.install.root}${/}bin${/}DefaultDB${/}-", "read,write,delete";
    // where, ${user.install.root} is the system property whose value is
    // located in the app_server_root directory.
 };
```

To use a `was.policy` file for your application, perform the following steps:

Procedure

1. Create a `was.policy` file using the policy tool. For more information on using the policy tool, see “Using PolicyTool to edit policy files for Java 2 security” on page 833.
2. Add the required permissions in the `was.policy` file using the policy tool.
3. Place the `was.policy` file in the application enterprise archive (EAR) file under the `META-INF` directory. Update the application EAR file with the newly created `was.policy` file by using the **jar** command.
4. Verify that the `was.policy` file is inserted and start an assembly tool.
5. Verify that the `was.policy` file in the application is syntactically correct. In an assembly tool, right-click the enterprise application module and click **Run Validation**.

Results

An application EAR file is now ready to run when Java 2 security is enabled.

Example

This step is required for applications to run properly when Java 2 security is enabled. If the `was.policy` file is not created and it does not contain required permissions, the application might not access system resources.

The symptom of the missing permissions is the java.security.AccessControlException exception. The missing permission is listed in the exception data, for example,

When an application program receives this exception and adding this permission is justified, include the permission in the `was.policy` file, for example,

The previous permission information lines are split for the illustration. Enter the permission on one line.

What to do next

Install the application.

Configuring static policy files in Java 2 security

By configuring the static policy files, the required permission will be granted for all of the Java programs.

Before you begin

Java 2 security uses several policy files to determine the granted permission for each Java program.

See the topic about Java 2 security policy files for the list of available policy files that are supported by WebSphere Application Server.

Two types of policy files are supported by WebSphere Application Server: dynamic policy files and static policy files. Static policy files provide the default permissions. Dynamic policy files provide application permissions.

Table 117. Policy Files. This table lists the policy files.

Policy file name	Description
`java.policy`	Contains default permissions for all of the Java programs on the node. This file seldom changes.
`server.policy`	Contains default permissions for all of the WebSphere Application Server programs on the node. This file is rarely updated.
`client.policy`	Contains default permissions for all of the applets and client containers on the node.

The static policy file is not a configuration file that is managed by the repository and the file replication service. Changes to this file are local and do not get replicated to the other machine.

Procedure

1. Identify the policy file to update.
   - If the permission is required only by an application, update the dynamic policy file. Refer to "Configuring Java 2 security policy files" on page 834.
   - If the permission is required only by applets and client containers, update the `client.policy` file. Refer to "client.policy file permissions" on page 851.
   - If the permission is required only by WebSphere Application Server (servers, agents, managers and application servers), update the `server.policy` file. Refer to "server.policy file permissions" on page 850.
   - If the permission is required by all of the Java programs running on the Java virtual machine (JVM), update the `java.policy` file. Refer to "java.policy file permissions."
2. Stop and restart WebSphere Application Server.

Results

The required permission is granted for all of the Java programs that run with the restarted JVM.

Example

If Java programs on a node require permissions, the policy file needs updating. If the Java program that required the permission is not part of an enterprise application, update the static policy file. The missing permission results in the creation of the java.security.AccessControlException exception. The missing permission is listed in the exception data.

For example:

```
java.security.AccessControlException: access denied (java.io.FilePermission
C:/WAS_HOME/lib/mail-impl.jar read)
```

When a Java program receives this exception and adding this permission is justified, add a permission to an adequate policy file.

For example:

```
grant codeBase "file:user_client_installed_location" {
  permission java.io.FilePermission
  "C:/WAS_HOME/lib/mail-impl.jar",
  "read";
};
```

To decide whether to add a permission, refer to Access control exception for Java 2 security.

java.policy file permissions:

Java 2 security uses several policy files to determine the granted permission for each Java program.

See Java 2 security policy files for the list of available policy files that are supported by WebSphere Application Server.

The java.policy file is a global default policy file that is shared by all of the Java programs that run in the Java virtual machine (JVM) on the node. A change to the java.policy file is local for the node. The default Java policy is picked up automatically. Syntax errors in the policy files cause the application server to fail. An updated java.policy file is applied to all the Java programs that run in all the JVMs on the local node. Restart the programs for the updates to take effect. Modifying this file is not recommended. If a specific change is required to some of the Java programs on a node and the java.policy file requires updating, carefully modify the java.policy file with the policy tool. For more information, see “Using PolicyTool to edit policy files for Java 2 security” on page 833.

Default permissions for the java.policy file

The java.policy file is not a configuration file that is managed by the repository and the file replication service. Changes to this file are local and do not get replicated to the other machine. The java.policy file that is supplied by WebSphere Application Server is located at install_root/java/jre/lib/security/java.policy. This file contains these default permissions.

```
// Standard extensions get all permissions by default
grant codeBase "file:${java.home}/lib/ext/*" {
        permission java.security.AllPermission;
};
// default permissions granted to all domains
grant {
        // Allows any thread to stop itself using the java.lang.Thread.stop()
        // method that takes no argument.
        // Note that this permission is granted by default only to remain
        // backwards compatible.
        // It is strongly recommended that you either remove this permission
        // from this policy file or further restrict it to code sources
        // that you specify, because Thread.stop() is potentially unsafe.
        // See "http://java.sun.com/notes" for more information.
        // permission java.lang.RuntimePermission "stopThread";

        // allows anyone to listen on un-privileged ports
        permission java.net.SocketPermission "localhost:1024-", "listen";

        // "standard" properties that can be read by anyone

        permission java.util.PropertyPermission "java.version", "read";
        permission java.util.PropertyPermission "java.vendor", "read";
        permission java.util.PropertyPermission "java.vendor.url", "read";
        permission java.util.PropertyPermission "java.class.version", "read";
        permission java.util.PropertyPermission "os.name", "read";
        permission java.util.PropertyPermission "os.version", "read";
        permission java.util.PropertyPermission "os.arch", "read";
        permission java.util.PropertyPermission "file.separator", "read";
        permission java.util.PropertyPermission "path.separator", "read";
        permission java.util.PropertyPermission "line.separator", "read";

        permission java.util.PropertyPermission "java.specification.version", "read";
        permission java.util.PropertyPermission "java.specification.vendor", "read";
        permission java.util.PropertyPermission "java.specification.name", "read";

        permission java.util.PropertyPermission "java.vm.specification.version","read";
        permission java.util.PropertyPermission "java.vm.specification.vendor","read";
        permission java.util.PropertyPermission "java.vm.specification.name", "read";
        permission java.util.PropertyPermission "java.vm.version", "read";
        permission java.util.PropertyPermission "java.vm.vendor", "read";
        permission java.util.PropertyPermission "java.vm.name", "read";
       };
```

If some Java programs on a node require permissions that are not defined as defaults in the java.policy file, consider updating the java.policy file. Most of the time, other policy files are updated instead of the java.policy file. The missing permission causes the creation of the , java.security.AccessControlException exception. The missing permission is listed in the exception data.

For example:

```
java.security.AccessControlException: access denied (java.io.FilePermission
C:\WebSphere\AppServer\java\jre\lib\ext\mail.jar read)
```

The previous two lines are one continuous line.

When a Java program receives this exception and adding this permission is justified, add a permission to the java.policy file.

For example:

```
grant codeBase "file:user_client_installed_location" {
permission java.io.FilePermission
"C:\WebSphere\AppServer\java\jre\lib\ext\mail.jar", "read"; };
```

To decide whether to add a permission, refer to Access control exception for Java 2 security.

Restart all of the Java processes for the updated `java.policy` file to take effect.

server.policy file permissions:

Java 2 security uses several policy files to determine the granted permission for each Java program.

See Java 2 security policy files for the list of available policy files that are supported by WebSphere Application Server.

The `server.policy` file is a default policy file that is shared by all of the WebSphere Application Servers on a node. The `server.policy` file is not a configuration file that is managed by the repository and the file replication service. Changes to this file are local and do not replicate to the other machine.

If the default permissions for a server (the union of the permissions that is defined in the `java.policy` file and the `server.policy` file) are enough, no action is required. The default server policy is picked up automatically. If a specific change is required to some of the server programs on a node, update the `server.policy` file with the Policy Tool. Refer to the “Using PolicyTool to edit policy files for Java 2 security” on page 833 topic to edit policy files. Changes to the `server.policy` file are local for the node. Syntax errors in the policy files cause the application server to fail. Edit these policy files carefully. An updated `server.policy` file is applied to all the server programs on the local node. Restart the servers for the updates to take effect.

If you want to add permissions to an application, use the `app.policy` file and the `was.policy` file.

Note: Updates to the `app.policy` file only apply to the enterprise applications on the node to which the `app.policy` file belongs.

When you do need to modify the `server.policy` file, locate this file at: *profile_root*`/properties/server.policy`. This file contains these default permissions:

```
// Allow to use sun tools
grant codeBase "file:${java.home}/../lib/tools.jar" {
  permission java.security.AllPermission;
};

// WebSphere system classes
grant codeBase "file:${was.install.root}/plugins/-" {
  permission java.security.AllPermission;
};
grant codeBase "file:${was.install.root}/lib/-" {
  permission java.security.AllPermission;
};
grant codeBase "file:${was.install.root}/classes/-" {
  permission java.security.AllPermission;
};

// Allow the WebSphere deploy tool all permissions
grant codeBase "file:${was.install.root}/deploytool/-" {
  permission java.security.AllPermission;
};

// Allow Channel Framework classes all permission
grant codeBase "file:${was.install.root}/installedChannels/-" {
  permission java.security.AllPermission;
};
```

```
// WebSphere optional runtime classes
grant codeBase "file:${was.install.root}/optionalLibraries/-" {
  permission java.security.AllPermission;
};
```

If some server programs on a node require permissions that are not defined as defaults in the `server.policy` file and the `server.policy` file, update the `server.policy` file. The missing permission creates the java.security.AccessControlException exception. The missing permission is listed in the exception data.

For example:

```
java.security.AccessControlException: access denied (java.io.FilePermission
C:\WebSphere\AppServer\java\jre\lib\ext\mail-impl.jar read)
```

The previous two lines are split into two lines for illustrative purposes only.

When a Java program receives this exception and adding this permission is justified, add a permission to the `server.policy` file.

For example:

```
grant codeBase "file:user_client_installed_location" {
permission java.io.FilePermission
"C:\WebSphere\AppServer\java\jre\lib\ext\mail.jar", "read"; };
```

To decide whether to add a permission, refer to Access control exception for Java 2 security.

Restart all of the Java processes for the updated `server.policy` file to take effect.

client.policy file permissions:

Java 2 security uses several policy files to determine the granted permission for each Java program.

For the list of available policy files that are supported by WebSphere Application Server, see Java 2 security policy files.

- The `client.policy` file is a default policy file that is shared by all of the WebSphere Application Server client containers and applets on a node.
- The union of the permissions that is contained in the `java.policy` file and the `client.policy` file are given to all of the client containers for WebSphere Application Server and applets running on the node.
- The `client.policy` file is not a configuration file that is managed by the repository and the file replication service. Changes to this file are local and do not replicate to the other machine.
- The `client.policy` file supplied by WebSphere Application Server is located in the *profile_root*/`properties/client.policy`.
- If the default permissions for a client (union of the permissions defined in the `java.policy` file and the `client.policy` file) are enough, no action is required. The default client policy is picked up automatically.
- If a specific change is required to some of the client containers and applets on a node, modify the `client.policy` file with the Policy Tool. Refer to “Using PolicyTool to edit policy files for Java 2 security” on page 833, to edit policy files. Changes to the `client.policy` file are local for the node.

This file contains these default permissions:

```
grant codeBase "file:${was.install.root}/java/ext/*" {
  permission java.security.AllPermission;
};

// JDK classes
grant codeBase "file:${was.install.root}/java/ext/-" {
  permission java.security.AllPermission;
};
```

```
grant codeBase "file:${was.install.root}/java/tools/ibmtools.jar" {
  permission java.security.AllPermission;
};
grant codeBase "file:/QIBM/ProdData/Java400/jdk14/lib/tools.jar" {
  permission java.security.AllPermission;
};

// WebSphere system classes
grant codeBase "file:${was.install.root}/lib/-" {
  permission java.security.AllPermission;
};
grant codeBase "file:${was.install.root}/plugins/-" {
  permission java.security.AllPermission;
};
grant codeBase "file:${was.install.root}/classes/-" {
  permission java.security.AllPermission;
};
grant codeBase "file:${was.install.root}/installedConnectors/-" {
  permission java.security.AllPermission;
};
grant codeBase "file:${user.install.root}/installedConnectors/-" {
  permission java.security.AllPermission;
};

grant codeBase "file:${was.install.root}/installedChannels/-" {
  permission java.security.AllPermission;
};

// J2EE 1.4 permissions for client container applications
// in $WAS_HOME/installedApps
grant codeBase "file:${user.install.root}/installedApps/-" {
  //Application client permissions
  permission java.awt.AWTPermission "accessClipboard";
  permission java.awt.AWTPermission "accessEventQueue";
  permission java.awt.AWTPermission "showWindowWithoutWarningBanner";
  permission java.lang.RuntimePermission "exitVM";
  permission java.lang.RuntimePermission "loadLibrary";
  permission java.lang.RuntimePermission "queuePrintJob";
  permission java.net.SocketPermission "*", "connect";
  permission java.net.SocketPermission "localhost:1024-", "accept,listen";
  permission java.io.FilePermission "*", "read,write";
  permission java.util.PropertyPermission "*", "read";

};

// J2EE 1.4 permissions for client container - expanded ear file code base
grant codeBase "file:${com.ibm.websphere.client.applicationclient.archivedir}/-" {
  permission java.awt.AWTPermission "accessClipboard";
  permission java.awt.AWTPermission "accessEventQueue";
  permission java.awt.AWTPermission "showWindowWithoutWarningBanner";
  permission java.lang.RuntimePermission "exitVM";
  permission java.lang.RuntimePermission "loadLibrary";
  permission java.lang.RuntimePermission "queuePrintJob";
  permission java.net.SocketPermission "*", "connect";
  permission java.net.SocketPermission "localhost:1024-", "accept,listen";
  permission java.io.FilePermission "*", "read,write";
  permission java.util.PropertyPermission "*", "read";
};
```

All of the client containers and applets on the local node are granted the updated permissions when they start. If some client containers or applets on a node require permissions that are not defined as defaults in the `java.policy` file and the default `client.policy` file, update the `client.policy` file. The missing permission creates the java.security.AccessControlException exception. The missing permission is listed in the exception data, for example,

```
java.security.AccessControlException: access denied (java.io.FilePermission
C:\WebSphere\AppServer\java\jre\lib\ext\mail.jar read)
```

The previous two lines of the example are one continuous line, but presented as such for illustrative purposes only.

When a client program receives this exception and adding this permission is justified, add a permission to the `client.policy` file, for example:

```
grant codebase "file:user_client_installed_location" {permission
 java.io.FilePermission "C:\WebSphere\AppServer\java\jre\lib\ext\mail.jar", "read"; };
```

To decide whether to add a permission, refer to Access control exception for Java 2 security.

If you update the policy file, you must restart the browser and any client applications.

Developing with programmatic security APIs for web applications

Use this information to programmatically secure APIs for web applications.

Before you begin

Programmatic security is used by security-aware applications when declarative security alone is not sufficient to express the security model of the application.

The authenticate, login, logout, getRemoteUser, isUserInRole and getAuthType servlet security methods are methods of the javax.servlet.http.HttpServletRequest interface. For more detailed information about these servlet security methods, read the Servlet security methods article.

Note:

> The logout, login, and authenticate APIs are new for Java Servlet 3.0 in this release of WebSphere Application Server.

You can configure several options for web authentication that determine how the web client interacts with protected and unprotected Uniform Resource Identifiers (URI). Also, you can specify whether WebSphere Application Server challenges the web client for basic authentication information if the certificate authentication for the HTTPS client fails. For more information, see the Selecting an authentication mechanism article.

When the isUserInRole method is used, declare a security-role-ref element in the deployment descriptor with a role-name subelement containing the role name that is passed to this method, or use the @DeclareRoles annotation. Because actual roles are created during the assembly stage of the application, you can use a logical role as the role name and provide enough hints to the assembler in the description of the security-role-ref element to link that role to the actual role. During assembly, the assembler creates a role-link subelement to link the role name to the actual role. Creation of a security-role-ref element is possible if an assembly tool, such as Rational Application Developer, is used. You also can create the security-role-ref element during assembly stage using an assembly tool.

Procedure

1. Add the required security methods in the servlet code.
2. Create a security-role-ref element with the **role-name** field. If a security-role-ref element is not created during development, make sure it is created during the assembly stage.

Results

A programmatically secured servlet application.

Example

These steps are required to secure an application programmatically. This action is particularly useful when a web application needs to access external resources and wants to control the access to external resources using its own authorization table (external-resource to remote-user mapping). In this case, use

the getUserPrincipal or the getRemoteUser methods to get the remote user, then the application can consult its own authorization table to perform authorization. The remote user information also can help retrieve the corresponding user information from an external source such as a database or from an enterprise bean. You can use the isUserInRole method in a similar way.

After development, you can create a security-role-ref element:

```
<security-role-ref>
   <description>Provide hints to assembler for linking this role
                name to an actual role here<\description>
   <role-name>Mgr<\role-name>
</security-role-ref>
```

During assembly, the assembler creates a role-link element:

```
<security-role-ref>
   <description>Hints provided by developer to map the role
                name to the role-link</description>
   <role-name>Mgr</role-name>
   <role-link>Manager</role-link>
</security-role-ref>
```

You can add programmatic servlet security methods inside any servlet doGet, doPost, doPut, and doDelete service methods. The following example depicts using a programmatic security API:

```
public void doGet(HttpServletRequest request,
HttpServletResponse response) {

   ....
   // to logoff the current user
  request.logout();

   // to login with a new user
   request.login("bob","bobpwd")

   // to get remote user using getUserPrincipal()
   java.security.Principal principal = request.getUserPrincipal();
   String remoteUser = principal.getName();

   // to get remote user using getRemoteUser()
   remoteUser = request.getRemoteUser();

   // to check if remote user is granted Mgr role
   boolean isMgr = request.isUserInRole("Mgr");

   // use the above information in any way as needed by
   // the application
   ....

}
```

You can programmatic login with a user ID and password inside any servlet doGet, doPost, doPut, and doDelete service methods. The following example depicts using a programmatic login/logout API:

```
public void doGet(HttpServletRequest request,
HttpServletResponse response) {

   ....
   // to logout the current user. If you are not already authenticate, then no need to call the logout() method.
   request.logout();

   // to login with a new user
   request.login("utle","mypwd")

   // the user utle subject now set on the thread and the LTPA SSO cookie is set in the response
   ....

}
```

You can programmatic authenticate with a different identity inside any servlet doGet, doPost, doPut, and doDelete service methods. In this example, if the web servlet is configured to use basicAuth, the web server returns a response code 401, the login prompt is displayed, and you can enter the user ID and password to authenticate. The following example depicts using a programmatic login/logout API:

```
public void doGet(HttpServletRequest request,
HttpServletResponse response) {

   ....
   // to logout the current user. If you are not already authenticate, then no need to call the logout() method.
```

```
   // to login with a new user
   request.authenticate(response);

   // the new user subject now set on the thread and the LTPA SSO cookie is set in the response
   ....

}
```

When developing Servlet 3.0 modules, the value of the rolename argument in isCallerInRole method can be defined using Java annotations instead of declaring a security-role-ref elements in the deployment descriptor.

```
@javax.annotation.security.DeclareRoles("Mgr")
public void doGet(HttpServletRequest request,
HttpServletResponse response) {

   ....

   // to get remote user using getUserPrincipal()
   java.security.Principal principal = request.getUserPrincipal();
   String remoteUser = principal.getName();

   // to get remote user using getRemoteUser()
   remoteUser = request.getRemoteUser();

   // to check if remote user is granted Mgr role
   boolean isMgr = request.isUserInRole("Mgr");

   // use the above information in any way as needed by
   // the application
   ....

}
```

The following example depicts a web application or servlet using the programmatic security model.

This example illustrates one use and not necessarily the only use of the programmatic security model. The application can use the information that is returned by the getUserPrincipal, isUserInRole, and the getRemoteUser methods in any other way that is meaningful to that application. Use the declarative security model whenever possible.

File : HelloServlet.java

```
public class HelloServlet extends javax.servlet.http.HttpServlet {

 public void doPost(
  javax.servlet.http.HttpServletRequest request,
  javax.servlet.http.HttpServletResponse response)
  throws javax.servlet.ServletException, java.io.IOException {
 }
public void doGet(
  javax.servlet.http.HttpServletRequest request,
  javax.servlet.http.HttpServletResponse response)
  throws javax.servlet.ServletException, java.io.IOException {

        String s = "Hello";


        // get remote user using getUserPrincipal()
        java.security.Principal principal = request.getUserPrincipal();
        String remoteUserName = "";
        if( principal != null )
          remoteUserName = principal.getName();
// get remote user using getRemoteUser()
        String remoteUser = request.getRemoteUser();

        // check if remote user is granted Mgr role
        boolean isMgr = request.isUserInRole("Mgr");

        // display Hello username for managers and bob.
        if ( isMgr || remoteUserName.equals("bob") )
            s = "Hello " + remoteUserName;

   String message =  "<html> \n" +
              "<head><title>Hello Servlet</title></head>\n" +
         "<body> /n +"
    "<h1> "  +s+ </h1>/n " +
  byte[] bytes = message.getBytes();

  // displays "Hello" for ordinary users
        // and displays "Hello username" for managers and "bob".
```

```
        response.getOutputStream().write(bytes);
 }

}
```

After developing the servlet, you can create a security role reference for the HelloServlet servlet as shown in the following example:

```
<security-role-ref>
     <description> </description>
     <role-name>Mgr</role-name>
</security-role-ref>
```

What to do next

After developing an application, use an assembly tool to create roles and to link the actual roles to role names in the security-role-ref elements. See the information about securing web applications using an assembly tool.

Servlet security methods

The authenticate, login, logout, getRemoteUser, isUserInRole and getAuthType servlet security methods are methods of the javax.servlet.http.HttpServletRequest interface.

authenticate

Note: The authenticate, login and logout servlet security methods are new for Java Servlet 3.0 in this release of WebSphere Application Server.

The authenticate method authenticates a user by using the WebSphere Application Server container login mechanism configured for the servlet context.

The syntax of the authenticate method is as follows:

```
boolean authenticate(HttpServletResponse response))
```

The previous example uses the following element:

response
The HttpServletResponse associated with the HttpServletRequest.

The authenticate method returns `true` when authentication has been established or authentication is successful.

The authenticate method returns `false` if authentication is incomplete and the underlying login mechanism has committed, in the response, the message and HTTP status code to be returned to the user.

A java.io.IOException occurs if an error occurs while writing the response.

A ServletException occurs if the authentication failed, and the caller is responsible for handling the error (for example, the underlying login mechanism did not establish the message and the HTTP status code to be returned to the user).

Note: When the authenticate method is called, be aware of the following:

- WebSphere Application Server returns `HTTP 401` code to a client.
- The method depends on the WebSphere Application Server container login mechanism that is configured for the servlet context. For example, if you have a form login defined for this servlet, it prompts for a user name and password. The client sends the user ID and password to WebSphere Application Server for authentication.

Important: Make sure that the authenticate method returns `true` before using the new subject to call another service. For example:

```
Boolean authResultTrue = req.authenticate(response);
  if (!authResultTrue) {
 return;
} else {
 // Use the new invocation subject to call other services.
       }
```

login

The login method authenticates a user to the WebSphere Application Server with a user ID and password. If authentication is successful, it creates a user subject on the thread and Lightweight Third Party Authentication (LTPA) cookies (if single sign-on (SSO) is enabled).

The syntax of the login method is as follows:

```
login(java.lang.String username, java.lang.String password)
```

The previous example uses the following elements:

username
The string value that corresponds to the login identifier of the user.

password
The password of the user.

A ServletException occurs if the configured login mechanism does not support username and password authentication, if an identity had already been authenticated (prior to the call to login), or if validation of the provided username and password fails.

Note: You can set the security custom property com.ibm.websphere.security.webAlwaysLogin to `true` and it will authenticate to the WebSphere application with the username and password, even if it is already authenticated.

For more information about modifying security custom properties, read the Modifying an existing custom property in a global security configuration or in a security domain configuration article.

Note: The login method always uses the user ID and password to authenticate to the WebSphere application server and even the SSO information that is present in the HttpServletRequest.

Note: The authenticate and login methods set the invocation subject to the new subject. If the caller subject is null, it then sets the caller subject to the new subject. If the caller subject is not null, then the caller subject is not set to the new subject.

Since the authenticate and login methods set the invocation subject to the new subject, the RunAs defined by the run-As attribute in deployment descriptor, security annotation or dynamic annotation is ignored.

logout

The logout method logs the user out of the WebSphere Application Server and invalidates the HTTP session. During this process, WebSphere Application Server completes the following processes:

- Clears the LTPA cookies if SSO is enabled
- Invalidates the HTTP session
- Removes the user from the authentication cache
- Removes the user subject from the thread
- Clears the caller and invocation subjects
- Sets the authentication type to `null`

After logging out, access to a protected web resource requires re-authentication and the getUserPrincipal, getRemoteUser and getAuthType methods return `null`.

The syntax of the logout method is as follows:

```
logout()
```

A ServletException occurs if the logout fails.

Audit event types for the authenticate, login and logout methods

To audit authenticate, login and logout methods, you must create or extend some audit event type files. These event type are not part of the default event type files.

Table 118. Audit event types for the authenticate, login, and logout methods.

The audit event types required for the authenticate, login, and logout methods are:

Method	Audit event name	Audit outcome of the event
authenticate/login	SECURITY_AUTHN	SUCCESS and or FAILURE
logout	SECURITY_AUTHN_TERMINATE	SUCCESS
logout	SECURITY_AUTHN_TERMINATE	FAILURE

isUserInRole

(String role name): Returns `true` if the remote user is granted the specified security role. If the remote user is not granted the specified role, or if no user is authenticated, it returns `false`.

getRemoteUser

The getRemoteUser method returns the login of the user that makes the request if the user has been authenticated. If the user has not been authenticated, the getRemoteUser method returns `null`.

getAuthType

The getAuthType method returns the name of the authentication scheme that is used to protect the servlet. If the servlet is not protected, the getAuthType method returns `null`.

The authentication schemes used are:

FORM when form-based authentication is used

BASIC
when basic authentication is used.

CLIENT_CERT
when client certificate authentication is used.

Note:

For both the getRemoteUser and getAuthType methods, the data that is returned depends upon whether security is enabled in the application server where the servlet is deployed. The following possibilities exist:

- If application security is enabled and a servlet is protected, then the getRemoteUser method returns the login and the getAuthType method returns the configured authentication scheme.
- If application security is not enabled, both methods return `null`.

Web authentication settings

Use this page to specify the web authentication settings that are associated with a web client.

To view this administrative console page, complete the following steps:

1. Click **Security** > **Global security**.
2. Under Authentication, expand `Web and SIP security` and click `General settings`.

You can override the global Web authentication settings that you select on this panel by specifying one or more of the following system properties on the server level. Complete the following steps to specify one of these system properties:

1. Click **Servers** > **Server Types** > **WebSphere application servers** > ***server_name***.
2. Under Server infrastructure, click **Java and Process Management** > **Process definition**.
3. Under Additional properties, click **Java Virtual Machine** > **Custom Properties** > **New**.

Table 119. Web authentication system property values. This table lists the web authentication system property values.

Property name	Value	Explanation
com.ibm.wsspi.security.web.webAuthReq	lazy	This value is equivalent to the `Authenticate only when the URI is protected` option. **Note:** You can set webAuthReq differently through the administrative console or scripting when using a global or a security domain, but the global level always takes precedence.
com.ibm.wsspi.security.web.webAuthReq	persisting	This value is equivalent to the `Use available authentication data when an unprotected URI is accessed` option.
com.ibm.wsspi.security.web.webAuthReq	always	This value is equivalent to the `Authenticate when any URI is accessed` option.
com.ibm.wsspi.security.web.failOverToBasicAuth	true	This value is equivalent to the `Default to basic authentication when certificate authentication for the HTTPS client fails` option.

Authenticate only when the URI is protected:

The application server challenges the web client to provide authentication data when the web client accesses a Uniform Resource Identifier (URI) that is protected by a Java 2 Platform, Enterprise Edition (J2EE) role. The authenticated identity is available only when the web client accesses a protected URI.

This option is the default J2EE web authentication behavior that is also available in previous releases of WebSphere Application Server.

Note: When you select this option, the administrative console login page is missing images. You might encounter the following error in the administrative console: “CWLAA6003: Could not display the portlet, the portlet may not be started. Check the error logs”.

The missing images and the error message are a side-effect of this option. The images do not display because the URIs for the images now need authentication, which requires you to log in. You can ignore this error message.

Information	Value
Default:	Enabled

Use available authentication data when an unprotected URI is accessed:

The web client can access validated authenticated data that it previously could not access. This option enables the web client to call the getRemoteUser, isUserInRole, and getUserPrincipal methods to retrieve an authenticated identity from an unprotected URI.

When you select this option with the `Authenticate only when the URI is protected` option, the web client can use authenticated data when the URI is protected or not protected.

When this option is selected and Form-based authentication is being used, a WASPostParam cookie is generated during the authentication procedure of the HTTP POST request even if the target URL is unprotected. A WASPOSTParam cookie is a temporary cookie used to store HTTP POST parameters. This results in the Web client being sent the unnecessary cookie with an HTTP response. This might cause unexpected behavior when the size of the cookie is larger than the browser limit. To avoid this behavior, a custom property, com.ibm.websphere.security.util.postParamMaxCookieSize can be set to cause the security code to stop generating the cookie if the maximum size is reached.

Important: This option does not challenge the web client to provide authenticated data if the web client accesses an unprotected URI without authenticated data.

Information	**Value**
Default:	Enabled

Authenticate when any URI is accessed:

The web client must provide authentication data regardless of whether the URI is protected.

Information	**Value**
Default:	Disabled

Default to basic authentication when certificate authentication for the HTTPS client fails:

When the required HTTPS client certificate authentication fails, the application server uses the basic authentication method to challenge the web client to provide a user ID and password.

The HTTP client certification authentication that is performed by the application server security is different from the client authentication that is performed by the web server plug-in. If you configure the web server plug-in for mutual authentication and client authentication fails, the following situations will occur:

- The web server produces a error and the web request is not processed by application server security.
- The application server cannot fail over to basic authentication.

Information	**Value**
Default:	Disabled

Developing with programmatic APIs for EJB applications

Use this topic to programmatically secure your Enterprise JavaBeans (EJB) applications.

About this task

Programmatic security is used by security-aware applications when declarative security alone is not sufficient to express the security model of the application. The javax.ejb.EJBContext application programming interface (API) provides two methods whereby the bean provider can access security information about the enterprise bean caller.

- **IsCallerInRole**(String rolename): Returns `true` if the bean caller is granted the security role that is specified by role name. If the caller is not granted the specified role, or if the caller is not authenticated, it returns `false`. If the specified role is granted Everyone access, it always returns `true`.
- **getCallerPrincipal**: Returns the java.security. Principal object that contains the bean caller name. If the caller is not authenticated, it returns a principal that contains an unauthorized name.

You can enable a login module to indicate which principal class is returned by these calls.

When the isCallerInRole method is used, declare a security-role-ref element in the deployment descriptor with a role-name that is subelement containing the role name that is passed to this method. Because actual roles are created during the assembly stage of the application, you can use a logical role as the role name and provide enough hints to the assembler in the description of the security-role-ref element to link that role to an actual role. During assembly, the assembler creates a role-link subelement to link the role-name to the actual role. Creation of a security-role-ref element is possible if an assembly tool such as Rational Application Developer is used. You also can create the security-role-ref element during the assembly stage using an assembly tool.

Procedure

1. Add the required security methods in the EJB module code.
2. Create a security-role-ref element with a role-name field for all the role names used in the isCallerInRole method. If a security-role-ref element is not created during development, make sure it is created during the assembly stage.

Results

Performing the previous steps result in a programmatically secured EJB application.

Example

Hard coding security policies in applications is strongly discouraged. The Java Platform, Enterprise Edition (Java EE) security model capabilities of declaratively specifying security policies is encouraged wherever possible. Use these APIs to develop security-aware EJB applications.

Using Java EE security model capabilities to specify security policies declaratively is useful when an EJB application wants to access external resources and wants to control the access to these external resources using its own authorization table (external-resource to user mapping). In this case, use the getCallerPrincipal method to get the caller identity and then the application can consult its own authorization table to perform authorization. The caller identification also can help retrieve the corresponding user information from an external source, such as database or from another enterprise bean. You can use the isCallerInRole method in a similar way.

After development, you can create a security-role-ref element:

```
<security-role-ref>
<description>Provide hints to assembler for linking this role-name to
actual role here<\description>
<role-name>Mgr<\role-name>
</security-role-ref>
```

During assembly, the assembler creates a role-link element:

```
<security-role-ref>
<description>Hints provided by developer to map role-name to role-link</description>
<role-name>Mgr</role-name>
<role-link>Manager</role-link>
</security-role-ref>
```

You can add programmatic EJB component security methods for example isCallerInRole and getCallerPrincipal, inside any business methods of an enterprise bean. The following example of programmatic security APIs includes a session bean:

```
public class aSessionBean implements SessionBean {

       .....

       // SessionContext extends EJBContext. If it is entity bean use EntityContext
       javax.ejb.SessionContext context;

       // The following method will be called by the EJB container
       // automatically
       public void setSessionContext(javax.ejb.SessionContext ctx) {
              context = ctx; // save the session bean's context
       }

       ....

       private  void aBusinessMethod()  {
       ....

       // to get  bean's caller using getCallerPrincipal()
       java.security.Principal principal = context.getCallerPrincipal();
       String  callerId= principal.getName();

       // to check if  bean's caller is granted Mgr role
       boolean isMgr = context.isCallerInRole("Mgr");

       // use the above information in any way as needed by the
       //application

       ....
       }

       ....
}
```

When developing EJB 3.x modules, the value of the rolename argument in isCallerInRole method can be defined using Java annotations instead of declaring a security-role-ref elements in the deployment descriptor.

```
  @javax.annotation.security.DeclareRoles("Mgr")
  @Stateless           // annotation is used to indicate a session bean
  public class aSessionBean implements MyBusinessInterface { //you don't have to extend sessionbean interface
        .....
        // SessionContext extends EJBContext. In EJB 3.0 use Resource annotation to inject context
    @Resource
        javax.ejb.SessionContext context;       }

       ....

       private  void aBusinessMethod()  {
       ....

       // to get  bean's caller using getCallerPrincipal()
       java.security.Principal principal = context.getCallerPrincipal();
       String  callerId= principal.getName();

       // to check if  bean's caller is granted Mgr role
       boolean isMgr = context.isCallerInRole("Mgr");

       // use the above information in any way as needed by the
       //application

       ....
       }

       ....
}
```

What to do next

After developing an application, use an assembly tool to create roles and to link the actual roles to role names in the security-role-ref elements. See the information about securing web applications using an assembly tool.

Example: Enterprise bean application code

The following Enterprise JavaBeans (EJB) component example illustrates the use of the isCallerInRole and the getCallerPrincipal methods in an EJB module.

Using declarative security is recommended. The following example is one way of using the isCallerInRole and the getCallerPrincipal methods. The application can use this result in any way that is suitable.

A remote interface

```
File : Hello.java

package tests;
import java.rmi.RemoteException;
/**
 * Remote interface for Enterprise Bean: Hello
 */
public interface Hello extends javax.ejb.EJBObject {
      public abstract String getMessage()throws RemoteException;
     public abstract void setMessage(String s)throws RemoteException;
}
```

A home interface

```
File : HelloHome.java
package tests;
/**
 * Home interface for Enterprise Bean: Hello
 */
public interface HelloHome extends javax.ejb.EJBHome {
 /**
  * Creates a default instance of Session Bean: Hello
  */
 public tests.Hello create() throws javax.ejb.CreateException,
  java.rmi.RemoteException;
}
```

A bean implementation

```
File : HelloBean.java

package tests;
/**
 * Bean implementation class for Enterprise Bean: Hello
 */
public class HelloBean implements javax.ejb.SessionBean {
 private javax.ejb.SessionContext mySessionCtx;
 /**
  * getSessionContext
  */
 public javax.ejb.SessionContext getSessionContext() {
  return mySessionCtx;
 }
 /**
  * setSessionContext
  */
 public void setSessionContext(javax.ejb.SessionContext ctx) {
  mySessionCtx = ctx;
 }
 /**
  * ejbActivate
  */
 public void ejbActivate() {
 }
```

```
 /**
  * ejbCreate
  */
 public void ejbCreate() throws javax.ejb.CreateException {
 }
 /**
  * ejbPassivate
  */
 public void ejbPassivate() {
 }
 /**
  * ejbRemove
  */
 public void ejbRemove() {
 }

 public java.lang.String message;


      //business methods

      // all users can call getMessage()
      public String getMessage() {
          return message;
      }

      // all users can call setMessage() but only few users can set new message.
      public void setMessage(String s) {

        // get  bean's caller using getCallerPrincipal()
        java.security.Principal principal = mySessionCtx.getCallerPrincipal();
        java.lang.String  callerId= principal.getName();

        // check if  bean's caller is granted Mgr role
        boolean isMgr = mySessionCtx.isCallerInRole("Mgr");

        // only set supplied message if caller is "bob" or caller is granted Mgr role
        if ( isMgr || callerId.equals("bob") )
            message = s;
        else
            message = "Hello";
      }

}
```

After the development of the entity bean, create a security role reference in the deployment descriptor under the session bean, Hello:

```
<security-role-ref>
     <description>Only Managers can call setMessage() on this bean (Hello)</description>
     <role-name>Mgr</role-name>
</security-role-ref>
```

For an explanation of how to create a <security-role-ref> element, see Securing enterprise bean applications. Use the information under Map security-role-ref and role-name to role-link to create the element.

Customizing web application login

You can create a form login page and an error page to authenticate a user.

Before you begin

A web client or a browser can authenticate a user to a Web server using one of the following mechanisms:

- **HTTP basic authentication**: A web server requests the Web client to authenticate and the web client passes a user ID and a password in the HTTP header.

- **HTTPS client authentication**: This mechanism requires a user (web client) to possess a public key certificate. The web client sends the certificate to a web server that requests the client certificates. This authentication mechanism is strong and uses the Hypertext Transfer Protocol with Secure Sockets Layer (HTTPS) protocol.
- **Form-based Authentication**: A developer controls the look and feel of the login screens using this authentication mechanism.

The Hypertext Transfer Protocol (HTTP) basic authentication transmits a user password from the web client to the web server in simple base64 encoding. Form-based authentication transmits a user password from the browser to the web server in plain text. Therefore, both HTTP basic authentication and form-based authentication are not very secure unless the HTTPS protocol is used.

The web application deployment descriptor contains information about which authentication mechanism to use. When form-based authentication is used, the deployment descriptor also contains entries for login and error pages. A login page can be either an HTML page or a JavaServer Pages (JSP) file. This login page is displayed on the web client side when a secured resource (servlet, JSP file, HTML page) is accessed from the application. On authentication failure, an error page is displayed. You can write login and error pages to suit the application needs and control the look and feel of these pages. During assembly of the application, an assembler can set the authentication mechanism for the application and set the login and error pages in the deployment descriptor.

Form login uses the servlet sendRedirect method, which has several implications for the user. The sendRedirect method is used twice during form login:
- The sendRedirect method initially displays the form login page in the web browser. It later redirects the web browser back to the originally requested protected page. The sendRedirect(String URL) method tells the web browser to use the HTTP GET request to get the page that is specified in the web address. If HTTP POST is the first request to a protected servlet or JavaServer Pages (JSP) file, and no previous authentication or login occurred, then HTTP POST is not delivered to the requested page. However, HTTP GET is delivered because form login uses the sendRedirect method, which behaves as an HTTP GET request that tries to display a requested page after a login occurs.
- Using HTTP POST, you might experience a scenario where an unprotected HTML form collects data from users and then posts this data to protected servlets or JSP files for processing, but the users are not logged in for the resource. To avoid this scenario, structure your web application or permissions so that users are forced to use a form login page before the application performs any HTTP POST actions to protected servlets or JSP files.

Procedure

1. Create a form login page with the required look and feel, including the required elements to perform form-based authentication.
2. Create an error page. You can program error pages to retry authentication or to display an appropriate error message.
3. Place the login page and error page in the web application archive (`.war`) file relative to the top directory. For example, if the login page is configured as `/login.html` in the deployment descriptor, place it in the top directory of the WAR file. An assembler can also perform this step using the assembly tool.
4. Create a form logout page and insert it to the application only when the web application requires a form-based authentication mechanism.

   By default the URL to the logout page should point to the host to which the request was made or its domain. Otherwise, a generic logout page is displayed. If you need to point this URL to a different host, then you need to set the com.ibm.websphere.security.logoutExitPageDomainList property in the `security.xml` file with a list of URLs that are allowed for the logout page. You can choose to allow any logout exit page to be used by setting the com.ibm.websphere.security.allowAnyLogoutExitPageHost property to a value of `true`. Setting this property to `true` might open your systems to a potential URL redirect attacks.

Example: Form login

You can use the WebSphere Application Server login facilities to implement and configure form login procedures. Use the following technologies for WebSphere Application Server and Java Platform, Enterprise Edition (Java EE) login functionality:

- Java EE form-based login
- Java EE servlet filter with login
- IBM extension: form-based login

The form login sample is part of the Technology Samples package. For more information on how to access the form login sample, see Accessing the samples.

Form login usage

For the authentication to proceed appropriately, the action of the login form must always have the j_security_check action. The following example shows how to code the form into the HTML page:

```
<form method="POST" action="j_security_check">
<input type="text" name="j_username">
<input type="text" name="j_password">
<\form>
```

Use the **j_username** input field to get the user name, and use the **j_password** input field to get the user password.

On receiving a request from a web client, the web server sends the configured form page to the client and preserves the original request. When the web server receives the completed form page from the web client, the server extracts the user name and password from the form and authenticates the user. On successful authentication, the web server redirects the call to the original request. If authentication fails, the web server redirects the call to the configured error page.

The following example depicts a login page in HTML (`login.html`):

```
<!DOCTYPE HTML PUBLIC "-//W3C/DTD HTML 4.0 Transitional//EN">
<html>
<META HTTP-EQUIV = "Pragma" CONTENT="no-cache">
<title> Security FVT Login Page </title>
<body>
<h2>Form Login</h2>
<FORM METHOD=POST ACTION="j_security_check">
<p>
<font size="2"> <strong> Enter user ID and password: </strong></font>
<BR>
<strong> User ID</strong> <input type="text" size="20" name="j_username">
<strong> Password </strong>  <input type="password" size="20" name="j_password">
<BR>
<BR>
<font size="2">  <strong> And then click this button: </strong></font>
<input type="submit" name="login" value="Login">
</p>

</form>
</body>
</html>
```

The following example depicts an error page in a JSP file:

```
<!DOCTYPE HTML PUBLIC "-//W3C/DTD HTML 4.0 Transitional//EN">
<html>
<head><title>A Form login authentication failure occurred</head></title>
<body>
<H1><B>A Form login authentication failure occurred</H1></B>
<P>Authentication may fail for one of many reasons. Some possibilities include:
<OL>
<LI>The user-id or password may be entered incorrectly; either misspelled or the
wrong case was used.
<LI>The user-id or password does not exist, has expired, or has been disabled.
</OL>
</P>
</body>
</html>
```

After an assembler configures the web application to use form-based authentication, the deployment descriptor contains the login configuration as shown:

```
<login-config id="LoginConfig_1">
<auth-method>FORM<auth-method>
<realm-name>Example Form-Based Authentication Area</realm-name>
<form-login-config id="FormLoginConfig_1">
<form-login-page>/login.html</form-login-page>
<form-error-page>/error.jsp</form-error-page>
</form-login-config>
</login-config>
```

A sample web application archive (WAR) file directory structure that shows login and error pages for the previous login configuration follows:

```
META-INF
     META-INF/MANIFEST.MF
     login.html
     error.jsp
     WEB-INF/
     WEB-INF/classes/
     WEB-INF/classes/aServlet.class
```

Form logout

Form logout is a mechanism to log out without having to close all Web-browser sessions. After logging out of the form logout mechanism, access to a protected web resource requires re-authentication. This feature is not required by J2EE specifications, but it is provided as an additional feature in WebSphere Application Server security.

Suppose that you want to log out after logging into a web application and perform some actions. A form logout works in the following manner:

1. The logout-form URI is specified in the web browser and loads the form.
2. The user clicks **Submit** on the form to log out.
3. The WebSphere Application Server security code logs the user out. During this process, the Application Server completes the following processes:
   a. Clears the Lightweight Third Party Authentication (LTPA) / single sign-on (SSO) cookies
   b. Invalidates the HTTP session
   c. Removes the user from the authentication cache
4. Upon logout, the user is redirected to a logout exit page.

Form logout does not require any attributes in a deployment descriptor. The form-logout page is an HTML or a JavaServer Pages (JSP) file that is included with the web application. The form-logout page is like most HTML forms except that like the form-login page, the form-logout page has a special post action. This post action is recognized by the web container, which dispatches the post action to a special internal form-logout servlet. The post action in the form-logout page must be `ibm_security_logout`.

You can specify a logout-exit page in the logout form and the exit page can represent an HTML or a JSP file within the same web application to which the user is redirected after logging out. Additionally, the logout-exit page permits a fully qualified URL in the form of `http://`*`hostname`*`:`*`port`*`/`*`URL`*. The logout-exit page is specified as a parameter in the form-logout page. If no logout-exit page is specified, a default logout HTML message is returned to the user.

Here is a sample form logout HTML form. This form configures the logout-exit page to redirect the user back to the login page after logout.

```
<!DOCTYPE HTML PUBliC "-//W3C/DTD HTML 4.0 Transitional//EN">
<html>
  <META HTTP-EQUIV = "Pragma" CONTENT="no-cache">
  <title>Logout Page </title>
  <body>
  <h2>Sample Form Logout</h2>
      <FORM METHOD=POST ACTION="ibm_security_logout" NAME="logout">
      <p>
      <BR>
      <BR>
      <font size="2"><strong> Click this button to log out: </strong></font>
      <input type="submit" name="logout" value="Logout">
```

```
      <INPUT TYPE="HIDDEN" name="logoutExitPage" VALUE="/login.html">
      </p>
      </form>
  </body>
</html>
```

What to do next

After developing login and error pages, add them to the Web application. Use the assembly tool to configure an authentication mechanism and insert the developed login page and error page in the deployment descriptor of the application.

Developing servlet filters for form login processing

You can control the look and feel of the login screen using the form-based login mechanism. In form-based login, you specify a login page that is used to retrieve the user ID and password information. You also can specify an error page that displays when authentication fails.

About this task

If additional authentication or additional processing is required before and after authentication, servlet filters are an option. Servlet filters can dynamically intercept requests and responses to transform or to use the information that is contained in the requests or responses. One or more servlet filters can be attached to a servlet or to a group of servlets. Servlet filters also can attach to JavaServer Pages (JSP) files and HTML pages. All of the attached servlet filters are called before the servlet is invoked.

Both form-based login and servlet filters are supported by any servlet Version 2.3 specification-complaint web container. The form login servlet performs the authentication and servlet filters perform additional authentication, auditing, or logging information.

To perform pre-login and post-login actions using servlet filters, configure these filters for either form login page support or for the /j_security_check URL. The j_security_check is posted by a form login page with the j_username parameter that contains the user name and the j_password parameter that contains the password. A servlet filter can use the user name parameter and password information to perform more authentication or other special needs.

Procedure

1. A servlet filter implements the javax.servlet.Filter class. Implement three methods in the filter class:
   - **init(javax.servlet.FilterConfig cfg)**. This method is called by the container once, when the servlet filter is placed into service. The FilterConfig passed to this method contains the init-parameters of the servlet filter. Specify the init-parameters for a servlet filter during configuration using the assembly tool.
   - **destroy**. This method is called by the container when the servlet filter is taken out of a service.
   - **doFilter(ServletRequest req, ServletResponse res, FilterChain chain)**. This method is called by the container for every servlet request that maps to this filter before invoking the servlet. The FilterChain chain that is passed to this method can be used to invoke the next filter in the chain of filters. The original requested servlet runs when the last filter in the chain calls the chain.doFilter method. Therefore, all filters call the chain.doFilter method for the original servlet to run after filtering. If an additional authentication check is implemented in the filter code and results in failure, the original servlet does not run. The chain.doFilter method is not called and can be redirected to some other error page.
2. If a servlet maps to many servlet filters, servlet filters are called in the order that is listed in the `web.xml` deployment descriptor of the application. Place the servlet filter class file in the `WEB-INF/classes` directory of the application.

Example

An example of a servlet filter.

This login filter can map to the /j_security_check URL to perform pre-login and post-login actions.

```
import javax.servlet.*;
     public class LoginFilter implements Filter {
     protected FilterConfig filterConfig;
     // Called once when this filter is instantiated.
     // If mapped to j_security_check, called
     // very first time j_security_check is invoked.
     public void init(FilterConfig filterConfig) throws ServletException {
        this.filterConfig = filterConfig;
        }
     public void destroy() {
        this.filterConfig = null;
        }
      // Called for every request that is mapped to this filter.
     // If mapped to j_security_check,
     // called for every  j_security_check action
     public void doFilter(ServletRequest request,
     ServletResponse response, FilterChain chain)
         throws java.io.IOException, ServletException   {
         // perform pre-login action here
         chain.doFilter(request, response);
         // calls the next filter in chain.
         // j_security_check if this filter is
         // mapped to j_security_check.
        // perform post-login action here.
                 }
       }
```

Using servlet filters to perform pre-login and post-login processing during form login

This example illustrates one way that the servlet filters can perform pre-login and post-login processing during form login.

```
Servlet filter source code: LoginFilter.java
/**
 * A servlet filter example: This example filters j_security_check and
 * performs pre-login action to determine if the user trying to log in
 * is in the revoked list. If the user is on the revoked list, an error is
 * sent back to the browser.
 *
 * This filter reads the revoked list file name from the FilterConfig
 * passed in the init() method. It reads the revoked user list file and
 * creates a revokedUsers list.
 *
 * When the doFilter method is called, the user logging in is checked
 * to make sure that the user is not on the revoked Users list.
 *
 */

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class LoginFilter implements Filter {

   protected FilterConfig filterConfig;

   java.util.List revokeList;


   /**
    * init() : init() method called when the filter is instantiated.
    * This filter is instantiated the first time j_security_check is
    * invoked for the application (When a protected servlet in the
    * application is accessed).
    */
   public void init(FilterConfig filterConfig) throws ServletException {
      this.filterConfig = filterConfig;

      // read revoked user list
      revokeList = new java.util.ArrayList();
      readConfig();
   }


   /**
    * destroy() : destroy() method called when the filter is taken
    * out of service.
    */
   public void destroy() {
```

```
      this.filterConfig = null;
      revokeList = null;
   }

   /**
    * doFilter() : doFilter() method called before the servlet to
    * which this filter is mapped is invoked. Since this filter is
    * mapped to j_security_check,this method is called before
    * j_security_check action is posted.
    */
   public void doFilter(ServletRequest request, ServletResponse response,
FilterChain chain) throws java.io.IOException, ServletException {


      HttpServletRequest req = (HttpServletRequest)request;
      HttpServletResponse res = (HttpServletResponse)response;

      // pre login action

      // get username
      String username = req.getParameter("j_username");

      // if user is in revoked list send error
      if ( revokeList.contains(username) ) {
      res.sendError(javax.servlet.http.HttpServletResponse.SC_UNAUTHORIZED);
      return;
      }

      // call next filter in the chain : let j_security_check authenticate
      // user
      chain.doFilter(request, response);

      // post login action

   }

   /**
    * readConfig() : Reads revoked user list file and creates a revoked
    * user list.
    */
   private void readConfig() {
      if ( filterConfig != null ) {

         // get the revoked user list file and open it.
         BufferedReader in;
         try {
               String filename = filterConfig.getInitParameter("RevokedUsers");
               in = new BufferedReader( new FileReader(filename));
         } catch ( FileNotFoundException fnfe) {
               return;
         }

         // read all the revoked users and add to revokeList.
         String userName;
         try {
               while ( (userName = in.readLine()) != null )
                   revokeList.add(userName);
         } catch (IOException ioe) {
         }

      }

   }


}
```

Important: In the previous code sample, the line that begins `public void doFilter(ServletRequest request` is broken into two lines for illustrative purposes only. The `public void doFilter(ServletRequest request` line and the line after it are one continuous line.

An example of the `web.xml` file that shows the LoginFilter filter configured and mapped to the j_security_check URL:

```
<filter id="Filter_1">
    <filter-name>LoginFilter</filter-name>
    <filter-class>LoginFilter</filter-class>
         <description>Performs pre-login and post-login operation</description>
    <init-param>
    <param-name>RevokedUsers</param-name>
    <param-value>c:\WebSphere\AppServer\installedApps\
                          <app-name>\revokedUsers.lst</param-value>
    </init-param>
</filter-id>
```

```
<filter-mapping>
 <filter-name>LoginFilter</filter-name>
     <url-pattern>/j_security_check</url-pattern>
</filter-mapping>
```

An example of a revoked user list file:

```
user1
cn=user1,o=ibm,c=us
user99
cn=user99,o=ibm,c=us
```

Configuring servlet filters for form login processing

IBM Rational Application Developer or an assembly tool can configure the servlet filters. Two steps are involved in configuring a servlet filter.

Procedure

1. Name the servlet filter and assign the corresponding implementation class to the servlet filter.

   Optionally, assign initialization parameters that get passed to the init method of the servlet filter.After configuring the servlet filter, the `web.xml` application deployment descriptor contains a servlet filter configuration similar to the following example:

```
<filter id="Filter_1">
   <filter-name>LoginFilter</filter-name>
   <filter-class>LoginFilter</filter-class>
   <description>Performs pre-login and post-login
        operation</description>
   <init-param>// optional
     <param-name>ParameterName</param-name>
     <param-value>ParameterName</param-value>
   </init-param>
</filter>
```

2. Map the servlet filter to a URL or a servlet.

   After mapping the servlet filter to a URL or a servlet, the `web.xml` application deployment descriptor contains servlet mapping similar to the following example:

```
<filter-mapping>
   <filter-name>LoginFilter</filter-name>
   <url-pattern>/j_security_check</url-pattern>
               // can be servlet <servlet>servletName</servlet>
</filter-mapping>
```

Example

You can use servlet filters to replace the CustomLoginServlet servlet, and to perform additional authentication, auditing, and logging.

The WebSphere Application Server Samples provide a form login sample that demonstrates how to use the WebSphere Application Server login facilities to implement and configure form login procedures. The sample integrates the following technologies to demonstrate the WebSphere Application Server and Java Platform, Enterprise Edition (Java EE) login functionality:

- Java EE form-based login
- Java EE servlet filter with login
- IBM extension: form-based login

The form login sample is part of the Technology Samples package.

Secure transports with JSSE and JCE programming interfaces

This topic provides detailed information about transport security using Java Secure Socket Extension (JSSE) and Java Cryptography Extension (JCE) programming interfaces. Within this topic, there is a description of the IBM version of the Java Cryptography Extension Federal Information Processing Standard (IBMJCEFIPS).

Java Secure Socket Extension

Java Secure Socket Extension (JSSE) provides the transport security for WebSphere Application Server. JSSE provides the application programming interface (API) framework and the implementation of the APIs for Secure Sockets Layer (SSL) and Transport Layer Security (TLS) protocols, including functionality for data encryption, message integrity, and authentication.

JSSE APIs are integrated into the Java 2 SDK, Standard Edition (J2SDK), Version 5. The API package for JSSE APIs is `javax.net.ssl.*`. Documentation for using JSSE APIs can be found in the J2SE 6 API documentation that is located at `http://java.sun.com/javase/6/docs/technotes/guides/security/jsse/JSSERefGuide.html`.

Several JSSE providers ship with the Java 2 SDK Version 5 that comes with WebSphere Application Server. The IBMJSSE provider is used in previous WebSphere Application Server releases. Associated with the IBMJSSE provider is the IBMJSSEFIPS provider, which is used when FIPS is enabled on the server. Both of these providers do not work with the Java Message Service (JMS) and HTTP transports in WebSphere Application Server Version 8.5. These transports take advantage of the J2SDK Verison 5 network input/output (NIO) asynchronous channels.

For more information on the new IBMJSSE2 provider, please review the documentation located at `http://www.ibm.com/developerworks/java/jdk/security/60/`.

Customizing Java Secure Socket Extension

You can customize a number of aspects of JSSE by plugging in different implementations of Cryptography Package Provider, X509Certificate and HTTPS protocols, or specifying different default keystore files, key manager factories, and trust manager factories. The following table summarizes which aspects can be customized, what the defaults are, and which mechanisms are used to provide customization.

Table 120. Customizable items. You can customize the following key aspects:

Customizable item	Default	How to customize
X509Certificate	X509Certificate implementation from IBM	The cert.provider.x509v1 security property
HTTPS protocol	Implementation from IBM	The java.protocol.handler.pkgs system property
Cryptography Package Provider	IBMJSSE2	A security.provider.n= line in security properties file. See description.
Default keystore	None	The * javax.net.ssl.keyStore system property
Default truststore	jssecacerts, if it exists. Otherwise, cacerts	The * javax.net.ssl.trustStore system property
Default key manager factory	IbmX509	The ssl.KeyManagerFactory.algorithm security property
Default trust manager factory	IbmX509	The ssl.TrustManagerFactory.algorithm security property

For aspects that you can customize by setting a system property, statically set the system property by using the `-D` option of the Java command. You can set the system property using the administrative console, or set the system property dynamically by calling the java.lang.System.setProperty method in your code: System.setProperty(propertyName,"propertyValue").

For aspects that you can customize by setting a Java security property, statically specify a security property value in the `java.security` properties file. The security property is propertyName=propertyValue. Dynamically set the Java security property by calling the java.security.Security.setProperty method in your code.

The `java.security` properties file is located in the following directory:

app_server_root`/java/jre/lib/security` directory.

Application Programming Interface

The JSSE provides a standard application programming interface (API) that is available in packages of the `javax.net` file, `javax.net.ssl` file, and the `javax.security.cert` file. The APIs cover:
- Sockets and SSL sockets
- Factories to create the sockets and SSL sockets
- Secure socket context that acts as a factory for secure socket factories
- Key and trust manager interfaces
- Secure HTTP URL connection classes
- Public key certificate API

You can find more information documented for the JSSE APIs if you access the following information:

Version 1.6

1. Access the http://www.ibm.com/developerworks/java/jdk/security/ website.
2. Click Java 1.6.
3. Click Javadoc HTML documentation in the Java Secure Socket Extension (JSSE) Guide section.

Samples using Java Secure Socket Extension

The Java Secure Socket Extension (JSSE) also provides samples to demonstrate its functionality. The Java Secure Socket Extension (JSSE) also provides samples to demonstrate its functionality. You can access the samples in the following location:

Version 1.6

1. Access the http://www.ibm.com/developerworks/java/jdk/security/ website.
2. Click Java 1.6.
3. Click `jssedocs_samples.zip` in the Java Secure Socket Extension (JSSE) Guide section.

Table 121. Extracted files. This table lists the following extracted files:

Files	Description
ClientJsse.java	Demonstrates a simple client and server interaction using JSSE. All enabled cipher suites are used.
OldServerJsse.java	Back-level samples
ServerPKCS12Jsse.java	Demonstrates a simple client and server interaction using JSSE with the PKCS12 keystore file. All enabled cipher suites are used.
ClientPKCS12Jsse.java	Demonstrates a simple client and server interaction using JSSE with the PKCS12 keystore file. All enabled cipher suites are used.
UseHttps.java	Demonstrates accessing an SSL or non-SSL web server using the Java protocol handler of the com.ibm.net.ssl.www.protocol class. The URL is specified with the `http` or `https` prefix. The HTML that is returned from this site is displayed.

See more instructions in the source code. Follow these instructions before you run the samples.

Permissions for Java 2 security

You might need the following permissions to run an application with JSSE: This list is for reference only.
- java.util.PropertyPermission "java.protocol.handler.pkgs", "write"
- java.lang.RuntimePermission "writeFileDescriptor"
- java.lang.RuntimePermission "readFileDescriptor"
- java.lang.RuntimePermission "accessClassInPackage.sun.security.x509"
- java.io.FilePermission "${user.install.root}${/}etc${/}.keystore", "read"
- java.io.FilePermission "${user.install.root}${/}etc${/}.truststore", "read"

For the IBMJSSE provider:
- java.security.SecurityPermission "putProviderProperty.IBMJSSE"
- java.security.SecurityPermission "insertProvider.IBMJSSE"

For the SUNJSSE provider:
- java.security.SecurityPermission "putProviderProperty.SunJSSE"
- java.security.SecurityPermission "insertProvider.SunJSSE"

Debugging

By configuring through the javax.net.debug system property, JSSE provides the following dynamic debug tracing: `-Djavax.net.debug=true`.

A value of `true` turns on the trace facility, provided that the debug version of JSSE is installed.

Documentation

See the Security: Resources for learning topic for documentation references to JSSE.

JCE

Java Cryptography Extension (JCE) provides cryptographic, key and hash algorithms for WebSphere Application Server. JCE provides a framework and implementations for encryption, key generation, key agreement, and Message Authentication Code (MAC) algorithms. Support for encryption includes symmetric, asymmetric, block and stream ciphers.

IBMJCE

The IBM version of the Java Cryptography Extension (IBMJCE) is an implementation of the JCE cryptographic service provider that is used in WebSphere Application Server. The IBMJCE is similar to SunJCE, except that the IBMJCE offers more algorithms:
- Cipher algorithm (AES, DES, TripleDES, PBEs, Blowfish, and so on)
- Signature algorithm (SHA1withRSA, MD5withRSA, SHA1withDSA)
- Message digest algorithm (MD5, MD2, SHA1, SHA-256, SHA-384, SHA-512)
- Message authentication code (HmacSHA1, HmacMD5)
- Key agreement algorithm (DiffieHellman)
- Random number generation algorithm (IBMSecureRandom, SHA1PRNG)
- Key store (JKS, JCEKS, PKCS12, JCERACFKS [z/OS only])

The IBMJCE belongs to the com.ibm.crypto.provider.* packages.

For further information, see the information on JCE on the following website: http://www.ibm.com/developerworks/java/jdk/security/60/.

IBMJCEFIPS

The IBM version of the Java Cryptography Extension Federal Information Processing Standard (IBMJCEFIPS) is an implementation of the JCE cryptographic service provider that is used in WebSphere Application Server. The IBMJCEFIPS service provider implements the following:
- Signature algorithms (SHA1withDSA, SHA1withRSA)
- Cipher algorithms (AES, TripleDES, RSA)
- Key agreement algorithm (DiffieHellman)
- Key (pair) generator (DSA, AES, TripleDES, HmacSHA1, RSA, DiffieHellman)
- Message authentication code (MAC) (HmacSHA1)
- Message digest (MD5, SHA-1, SHA-256, SHA-384, SHA-512)
- Algorithm parameter generator (DiffieHellman, DSA)
- Algorithm parameter (AES, DiffieHellman, DES, TripleDES, DSA)
- Key factory (DiffieHellman, DSA, RSA)

- Secret key factory (AES, TripleDES)
- Certificate (X.509)
- Secure random (IBMSecureRandom)

Application Programming Interface

Java Cryptography Extension (JCE) has a provider-based architecture. Providers can be plugged into the JCE framework by implementing the APIs that are defined by the JCE. The JCE APIs cover:
- Symmetric bulk encryption, such as DES, RC2, and IDEA
- Symmetric stream encryption, such as RC4
- Asymmetric encryption, such as RSA
- Password-based encryption (PBE)
- Key agreement
- Message authentication codes

There is more information documented for the JCE APIs on the `http://www.ibm.com/developerworks/java/jdk/security/` website.

Samples using Java Cryptography Extension

There are samples located on the `http://www.ibm.com/developerworks/java/jdk/security/` website in the `jceDocs_samples.zip` file. Unzip the file and locate the following samples in the `jceDocs/samples` directory:

Table 122. Samples using Java Cryptography Extension. This table describes samples using Java Cryptography Extension.

File	Description
SampleDSASignature.java	Demonstrates how to generate a pair of DSA keys (a public key and a private key) and use the key to digitally sign a message using the SHA1withDSA algorithm
SampleMarsCrypto.java	Demonstrates how to generate a Mars secret key, and how to do Mars encryption and decryption
SampleMessageDigests.java	Demonstrates how to use the message digest for MD2 and MD5 algorithms
SampleRSACrypto.java	Demonstrates how to generate an RSA key pair, and how to do RSA encryption and decryption
SampleRSASignatures.java	Demonstrates how to generate a pair of RSA keys (a public key and a private key) and use the key to digitally sign a message using the SHA1withRSA algorithm
SampleX509Verification.java	Demonstrates how to verify X509 certificates

Documentation

Refer to the Security: Resources for learning topic for documentation on JCE.

Configuring Federal Information Processing Standard Java Secure Socket Extension files

Use this topic to configure Federal Information Processing Standard Java Secure Socket Extension files.

About this task

In WebSphere Application Server, the Java Secure Socket Extension (JSSE) provider used is the IBMJSSE2 provider. This provider delegates encryption and signature functions to the Java Cryptography Extension (JCE) provider. Consequently, IBMJSSE2 does not need to be Federal Information Processing Standard (FIPS)-approved because it does not perform cryptography. However, the JCE provider requires FIPS-approval.

WebSphere Application Server provides a FIPS-approved IBMJCEFIPS provider that IBMJSSE2 can utilize. The IBMJCEFIPS provider that is shipped in WebSphere Application Server Version 8.5 supports the following SSL ciphers:

- SSL_RSA_WITH_AES_128_CBC_SHA
- SSL_RSA_WITH_3DES_EDE_CBC_SHA
- SSL_RSA_FIPS_WITH_3DES_EDE_CBC_SHA
- SSL_DHE_RSA_WITH_AES_128_CBC_SHA
- SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA
- SSL_DHE_DSS_WITH_AES_128_CBC_SHA
- SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA

Even though the IBMJSSEFIPS provider is still present, the runtime does not use this provider. If IBMJSSEFIPS is specified as a contextProvider, WebSphere Application Server automatically defaults to the IBMJSSE2 provider (with the IBMJCEFIPS provider) for supporting FIPS. When enabling the **Use the United States Federal Information Processing Standard (FIPS) algorithms** option on the server SSL certificate and key management panel, the runtime always uses IBMJSSE2, despite the contextProvider that you specify for SSL (IBMJSSE, IBMJSSE2 or IBMJSSEFIPS). Also, because FIPS requires the SSL protocol be TLS, the runtime always uses TLS when FIPS is enabled, regardless of the SSL protocol setting in the SSL repertoire. This simplifies the FIPS configuration in Version 8.5 because an administrator needs to enable only the **Use the United States Federal Information Processing Standard (FIPS) algorithms** option on the server SSL certificate and key management panel to enable all transports using SSL.

Procedure

1. Click **Security > SSL certificate and key management > Manage FIPS**.
2. Select the **Enable FIPS 140-2** option and click **Apply**. This option makes IBMJSSE2 and IBMJCEFIPS the active providers.
3. Accommodate Java clients that must access enterprise beans.

   Change the `com.ibm.security.useFIPS` property value from `false` to `true` in the *profile_root*`/properties/ssl.client.props` file.
4. Ensure that the com.ibm.ssl.protocol property within the *profile_root*`/properties/ssl.client.props` file is set to TLS.
5. Ensure that the `java.security` file includes the provider.

What to do next

After completing these steps, a FIPS-approved JSSE or JCE provider offers increased encryption capabilities. However, when you use FIPS-approved providers:

- By default, Microsoft Internet Explorer might not have TLS enabled. To enable TLS, open the Internet Explorer browser and click **Tools > Internet Options**. On the Advanced tab, select the Use TLS 1.0 option.

  Note: Netscape Version 4.7.*x* and earlier versions might not support TLS.
- If you have an administrative client that uses a SOAP connector and you enable FIPS, add the following line to the *profile_root*`/properties/soap.client.props` file:

  `com.ibm.ssl.contextProvider=IBMJSSEFIPS`
- When you select the **Use the Federal Information Processing Standard (FIPS)** option on the SSL certificate and key management panel, the Lightweight Third-Party Authentication (LTPA) token format is not backwards-compatible with previous releases of WebSphere Application Server. However, you can import the LTPA keys from a previous version of the application server.

Attention: The following error might occur when you attempt to stop WebSphere Application Server after enabling the FIPS option:

```
ADMU3007E: Exception com.ibm.websphere.management.exception.ConnectorException
```

Uncomment the following entry in the java.security file if it was previously removed or commented out, then restart the server:

```
security.provider.2=com.ibm.crypto.provider.IBMJCE
```

Note: When enabling FIPS, you cannot configure cryptographic token devices in the SSL repertoires. IBMJSSE2 must use IBMJCEFIPS when utilizing cryptographic services for FIPS.

The following FIPS 140-2 approved cryptographic providers that are the only devices that are supported with the FIPS option:

- IBMJCEFIPS (certificate 376)
- IBM Cryptography for C (ICC) (certificate 384)

The relevant certificates are listed on the NIST website: Cryptographic Module Validation Program FIPS 140-1 and FIPS 140-2 Pre-validation List

To unconfigure the FIPS provider, reverse the changes that you made in the previous steps. After you reverse the changes, verify that you have made the following changes to the `sas.client.props`, `soap.client.props`, and `java.security` files:

- In the `ssl.client.props` file, you must change the `com.ibm.security.useFIPS` value to `false`.
- In the `java.security` file, you must change the FIPS provider to a non-FIPS provider.

  If you are using the IBM SDK `java.security` file, you must change the first provider to a non-FIPS provider as shown in the following example:

```
#security.provider.1=com.ibm.crypto.fips.provider.IBMJCEFIPS
security.provider.1=com.ibm.crypto.provider.IBMJCE
security.provider.2=com.ibm.jsse.IBMJSSEProvider
security.provider.3=com.ibm.jsse2.IBMJSSEProvider2
security.provider.4=com.ibm.security.jgss.IBMJGSSProvider
security.provider.5=com.ibm.security.cert.IBMCertPath
#security.provider.6=com.ibm.crypto.pkcs11.provider.IBMPKCS11
```

  If you are using the Sun JDK `java.security` file, you must change the third provider to a non-FIPS provider as shown in the following example:

```
security.provider.1=sun.security.provider.Sun
security.provider.2=com.ibm.security.jgss.IBMJGSSProvider
security.provider.3=com.ibm.crypto.fips.provider.IBMJCEFIPS
security.provider.4=com.ibm.crypto.provider.IBMJCE
security.provider.5=com.ibm.jsse.IBMJSSEProvider
security.provider.6=com.ibm.jsse2.IBMJSSEProvider2
security.provider.7=com.ibm.security.cert.IBMCertPath
#security.provider.8=com.ibm.crypto.pkcs11.provider.IBMPKCS11
```

WebSphere Application Server security standards configurations

WebSphere Application Server can be configured to work with various security standards, which are typically used to meet security requirements required by the government.

Note: WebSphere Application Server integrates cryptographic modules, which include Java Secure Socket Extension (JSSE) and Java Cryptography Extension (JCE). Most of the requirements in the standards are handled in the JSSE and JCE, which must undergo the certification process to meet government standards. WebSphere Application Server must be configured to run with the JSSE and JCE enabled for a particular standard, and now supports the FIPS 140-2, SP800-131 and Suite B security standards.

- **FIPS 140-2** are Federal Information Processing Standards (FIPS) that specify requirements on cryptographic modules. WebSphere Application Server has been able to configure using this standard the longest. Many users can be configured to use this level, but might be required to move up to the newer SP800-131 or Suite B standard.

See The National Institute of Standards and Technology web site for more information about the 140-2 standard.

To configure FIPS 140-2, see the topic “Configuring Federal Information Processing Standard Java Secure Socket Extension files”.

- **SP800-131** is a requirement originated by the National Institute of Standards and Technology (NIST) which requires longer key lengths and stronger cryptography. The specification also provides a transition configuration to enable users to move to a strict enforcement of SP800-131. The transition configuration also enables users to run with a mixture of settings from both FIPS140-2 and SP800-131. SP800-131 can be run in two modes, `transition` and `strict`.

  Strict enforcement of SP800-131 requirements on WebSphere Application server includes the following:

  – The use of the TLSv1.2 protocol for the Secure Sockets Layer (SSL) context.
  – Certificates must have a minimum length of 2048. Elliptical Curve (EC) certificate require a minimum size of 244-bit curves.
  – Certificates must be signed with a signature algorithm of SHA256, SHA384, or SHA512. Valid signatureAlgorithms include:
    - SHA256withRSA
    - SHA384withRSA
    - SHA512withRSA
    - SHA256withECDSA
    - SHA384withECDSA
    - SHA512withECDSA
  – SP800-131 approved Cipher suites

  See The National Institute of Standards and Technology web site for more details about the SP800-131 standard.

  See the topic “Transitioning WebSphere Application Server to the SP800-131 security standard” for information on how to transition WebSphere Application Server to the SP800-131 strict standard. See the topic “Configuring WebSphere Application Server for SP800-131 standard strict mode” for information on how to configure SP800-131.

- **Suite B** is a requirement originated by the National Security Agency (NSA) to specify a cryptographic interoperability strategy. This standard is similar to SP800-131 with some tighter restrictions. Suite B can run in 2 modes: 128-bit or 192-bit. If using 192-bit mode, users must apply the unrestricted policy file to the JDK so that the stronger cipher required for the 192-bit mode can be used.

  See the topic "Configuring WebSphere Application Server for the Suite B security standard" for information on to configure Suite B.

  Suite B requirements on WebSphere Application Server includes the following:

  – The use of the TLSv1.2 protocol for the SSL Context
  – Suite B approved Cipher suites
  – Certificates:
    - 128 bit mode certificates must be signed with SHA256withECDSA
    - 192 bit mode certificates must be signed with SHA384withECDSA
  – Ciphers:
    - SSL_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
    - SSL_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384.

Properties used to enable the Security Standards

The IBM virtual machine for Java (JVM) runs in a given security mode based on system properties. WebSphere Application Server sets these system properties based on security configuration settings. The

security configuration can be set up through the administrative console or through scripting admin tasks. If an application sets these properties directly it can affect WebSphere Application Server SSL communication.

Table 123. JVM system properties to enable the security standard

Security standard	System property to enable	Valid values
FIPS 140-2	com.ibm.jsse2.usefipsprovider	`true` or `false`
SP800-131	com.ibm.jsse2.sp800-131	`transition` or `strict`
Suite B	com.ibm.jsse2.suiteB	128 or 192

WebSphere Application Server configuration clears out all of these properties if they are set, then sets them to how the security configuration is specified. WebSphere Application Server enables the security standard based on the custom properties set in the security configuration.

WebSphere Application Server security custom properties to enable the security standard

Table 124. WebSphere Application Server security custom properties to enable the security standard

Security standard	Security custom properties	JVM system property
FIPS 140-2	com.ibm.security.useFips=true com.ibm.websphere.security.FIPSLevel=FIPS140-2	com.ibm.jsse2.usefipsprovider=true
SP800-131- transition	com.ibm.security.useFips=true com.ibm.websphere.security.FIPSLevel=transition	com.ibm.jsse2.sp800-131=transition
SP800-131 – strict	com.ibm.security.useFips=true com.ibm.websphere.security.FIPSLevel=SP800-131	com.ibm.jsse2.sp800-131=strict
Suite B 128	com.ibm.security.useFips=true com.ibm.websphere.security.suiteB=128	com.ibm.jsse2.suiteB=128
Suite B 192	com.ibm.security.useFips=true com.ibm.websphere.security.suiteB=192	com.ibm.jsse2.suiteB=192

Convert certificates

Use this page to convert certificates to the selected security standard. All certificates in keystores associated with an Secure Socket Layer (SSL) configuration are converted.

To view this administrative console page, click **Security > SSL certificate and key management > Manage FIPS > Convert certificates**.

Algorithm

Specifies the signature algorithm used to convert the certificate to the selected security standard.

The following choices are available:

Strict Select for the strict enforcement of the SP800-131 standard.

Strict enforcement of SP800-131 requirements on WebSphere Application Server includes the following:

- The use of the TLSv1.2 protocol for the Secure Sockets Layer (SSL) context.
- Certificates must have a minimum length of 2048. Elliptical Curve (EC) certificate require a minimum size of 244-bit curves.
- Certificates must be signed with a signature algorithm of SHA256, SHA384, or SHA512. Valid signatureAlgorithms include:
  - SHA256withRSA
  - SHA384withRSA
  - SHA512withRSA

  - SHA256withECDSA
  - SHA384withECDSA
  - SHA512withECDSA
- SP800-131 approved Cipher suites

Suite B with 128 bit keys
This requirement places some tighter restrictions on the SP800-131 specification. 128-bit mode certificates must be signed with SHA256withECDSA.

Suite B with 192 bit keys
192 bit mode certificates must be signed with SHA384withECDSA.

To run in 192-bit mode, the unrestricted policy files must be in place on the JDK.

New certificate key size
Specifies the key size to use when converting the certificates.

The valid values are 512, 1024, 2048, 4096 and 8192. The default value is 2048.

Note: Elliptical Curve signature algorithms require specific sizes, so you must provide a size.

Certificates that can not be converted
Lists the certificates that are not compliant with the specified security standard and cannot be converted.

If certificates show up listed in this box, the server is unable to convert the certificates for you. You must replace these certificates with ones that meet Suite B requirements. Reasons why the server cannot convert the certificates might include:

- The certificate was created by a Certificate Authority (CA).
- The certificate is in a read-only keystore.

Manage FIPS
Use this page to disable Federal Information Processing Standards (FIPS) or to enable security standards that are required by the government.

WebSphere Application Server integrates cryptographic modules, which include Java Secure Socket Extension (JSSE) and Java Cryptography Extension (JCE). JSSE and JCE must undergo the certification process to meet government standards, and WebSphere Application Server must be configured to use them as specified by the standards.

To view this administrative console page, click **Security > SSL certificate and key management > Manage FIPS**.

Disable FIPS
Select to disable FIPS, which is the default.

Data type:	Default:	Range:
Boolean	Enabled	Enabled or Disabled

Enable FIPS 140-2
Select to enable FIPS 140-2. This option makes IBMJSSE2 and IBMJCEFIPS the active providers.

Federal Information Processing Standards (FIPS) specifies requirements on cryptographic modules. WebSphere Application Server has been able to configure using the FIPS 140-2 standard the longest. Many users can be configured to use this level, but might be required to move up to the newer SP800-131 or Suite B standard.

Data type:	Default:	Range:
Boolean	Enabled	Enabled or Disabled

Enable SP800-131

Select to enable SP800-131.

SP800-131 is a requirement originated by the National Institute of Standards and Technology (NIST) which requires longer key lengths and stronger cryptography. The specification also provides a transition configuration to enable users to move to a strict enforcement of SP800-131. The transition configuration also enables users to run with a mixture of settings from both FIPS140-2 and SP800-131. SP800-131 can be run in two modes, `transition` and `strict`.

Data type:	Default:	Range:
Boolean	Enabled	Enabled or Disabled

Enable Suite B: Accept 128 bit keys

Select to specify that suite B cryptography is used, and is configured to accept a 128-bit key size. Keystore certificate algorithms require Elliptical curve (EC) cryptography.

Data type:	Default:	Range:
Boolean	Enabled	Enabled or Disabled

Enable Suite B: Accept 192-bit keys

Select to specify that suite B cryptography is used, and is configured to accept a 192-bit key size. Keystore certificate algorithms require Elliptical curve (EC) cryptography.

Suite B can run in 2 modes: 128-bit or 192-bit. If using 192-bit mode, you must apply the unrestricted policy file to the JDK so that the stronger cipher required for the 192-bit mode can be used.

Data type:	Default:	Range:
Boolean	Enabled	Enabled or Disabled

Convert certificates

Select to convert certificates to the selected security standard. All certificates in keystores associated with an Secure Socket Layer (SSL) configuration are converted.

Configuring WebSphere Application Server for the Suite B security standard

You can configure WebSphere Application Server to use the new Suite B security standard.

Before you begin

Read the “WebSphere Application Server security standards configurations” topic for more background information regarding security standards.

About this task

The National Security Agency (NSA) created a cryptographic interoperability strategy called Suite B. It places specific requirements on the National Institute of Standards and Technology (NIST) SP800-131 standard.

Suite B requirements:

WebSphere Application Server must be compliant with the following Suite B requirements:

- SSL configuration must use the TLSv1.2 protocol.
- The com.ibm.jsse.suiteb system property must be set to 128 or 192.
- Certificates running in 128-bit mode must be created with the SHA256withECDSA signature algorithm. Certificates running in 192-bit mode must be created with the SHA384withECDSA signature algorithm.

  Note: To run in 192-bit mode, the unrestricted policy files must be in place on the JDK.
- Suite B approved cipher suites must be used.

To configure the server for the Suite B standard:

Procedure

1. Click **Security > SSL certificate and key management > Manage FIPS** To run in a Suite B mode, all of the certificates used for SSL on the server must be converted to certificates that comply with Suite B requirements.
2. To convert certificates, under Related Items click **Convert Certificates**.
3. Select the radio button labeled **128-bit or 192-bit** in the Algorithm box.

   Note: Elliptical Curve signature algorithms require specific sizes, so you must provide a size.
4. Click **Apply/Save**. If no certificates show up in the box labeled **Certificates that can not be converted**, then you can enable the standard.

   If certificates show up listed in the box labeled **Certificates that can not be converted**, the server is unable to convert the certificates for you. You must replace these certificates with ones that meet Suite B requirements. Reasons why the server cannot convert the certificates might include:
   - The certificate was created by a Certificate Authority (CA).
   - The certificate is in a read-only keystore.

   After certificates are converted to meet the Suite B specifications, follow the remaining steps to enable the Suite B standard.
5. Click **SSL certificate and key management > Manage FIPS**.
6. Select the **Suite B: Accept 128 bit key** for 128-bit mode or the **Suite B: Accept 192 bit key** for 192-bit mode.
7. Click **Apply/Save**.
8. Restart the servers and manually sync the nodes for the Suite B standard to take effect.

   When these changes are applied and the server is restarted, the SSL configurations on the server is modified to use the TLSv1.2 protocol, and the com.ibm.jsse.suiteb system property is set to the desired Suite B mode. The SSL configuration uses the appropriate SSL ciphers for the standard.

   There are wsadmin tasks also available that can enable the Suite B standard using scripting. :
   - Check the status of certificates for the security standard by using the listCertStatusForSecurityStandard task.
   - Convert certificates for the security standard by using the convertCertForSecurityStandard task.
   - Enable the security standard by using the enableFips task.
   - To see the security standard setting, use the getFipsInfo task.
9. Once the server is configured for SP800-131 strict mode, the `ssl.client.props` file must be modified so that administrative clients are running in SP800-131 strict mode. They are unable to make a SSL connection to the server with the change. Edit the `ssl.client.props` file by doing the following:
   a. Modify com.ibm.security.useFIPS to be set to `true`.
   b. Add the com.ibm.jsse.suiteb property, and set it to 128 or 192.
   c. Change the com.ibm.ssl.protocol property to TLSv1.2.

What to do next

The Suite B standard requires that the SSL connection use the TLSv1.2 protocol. For a browser to access the administrative console or an application, the browser must support and first be configured to use the TLSv1.2 protocol.

Note: When enabling the security standards on a Network Deployed, the node and deployment manager can be in an incompatible protocol state. Since configuring the security standard requires the server to be restarted, it is recommended that all node agents and servers be stopped, leaving the deployment manager running. Once the configuration changes are made through the console, restart the deployment manager.

Manually sync the nodes with syncNode, and start the node agents and servers. To use syncNode, you might need to update the `ssl.client.props` file to communicate with the deployment manager.

Transitioning WebSphere Application Server to the SP800-131 security standard

The National Institute of Standards and Technology (NIST) Special Publications 800-131 standard strengthens algorithms and increases the key lengths to improve security. The standard also provides for a transition period to move to the new standard. You can configure WebSphere Application Server for SP800-131 standard transition mode.

Before you begin

Read the "WebSphere Application Server security standards configurations" topic for more background information regarding security standards.

About this task

The transition period enables a user to run in a mixed environment of settings not supported under the standard along with those that are supported. The NIST SP800-131 standard requires that users be configured for strict enforcement of the standard by a specific timeframe. See The National Institute of Standards and Technology web site for more details.

The transition options can be very useful when trying to get to `strict` SP800-131. The servers can accept a mixture old settings and new requirements. For example. they can convert certificates but continue to use the TLS protocol.

WebSphere Application Server can be configured to run SP800-131 in a `transition` mode or a `strict` mode. For information on how to configure strict mode. read the Configuring WebSphere Application Server for strict mode SP800-131 security standard topic.

To run in the SP800-131 transition mode, the server must be in a specific configuration setting as well. Other strict requirements can be include as wanted.

- The com.ibm.jsse2.sp800-131 system property must be set to `transition` for the JSSE to run in the transition mode.
- The SSL configuration protocols must be one of the TLS settings. Valid values include TLS, TLSv1, TLSv1.1, and TLSv1.2.

Procedure

1. Click **Security > SSL certificate and key management > Manage FIPS**.
2. Select the **Enable SP800-131** radio button.
3. Select the **Transition** radio button.

4. You have the choice to change the protocols in SSL configuration to TLSv1.2 by optionally selecting the **Update the SSL configuration to require TLSv1.2** box. If you do not select this box, all SSL configurations are set to TLS.
5. Click **Apply/Save**.
6. Restart the servers.

   When these changes are applied, and the server is restarted, all of the SSL configuration on the server are modified to use the TLS or TLSv1.2 protocol, and the com.ibm.jsse2.sp800-131 system property is set to `transition`. The SSL configuration uses the appropriate SSL ciphers for the standard.

   Before you can move to the strict mode certificate, the protocol in the configuration must meet the strict requirements.

   You can go to directly to the SSL configuration and set protocols to TLSv1.2 by doing the following:
7. Click **Security > SSL certificate and key management > SSL Configurations**.
8. Select a SSL configuration from the collection panel.
9. Under Related Items, select **Quality of protection (QoP)**.
10. Select **TLSv1.2** from the pull-down box labeled Protocol
11. Click **Apply/Save**. To change the SSL protocol using scripting, the modifySSLConfig task can also be used.

    Certificates must have a minimum size of 2048 (244 if an Elliptical Curve certificate), and signed with SHA256, SHA384, or SHA512. You can create new ones on the console and replace the old one, or import certificates that meet the standards requirements.

    There are a number of options you can use to replace certificates.

    - Use the Convert Certificate panel. This panel converts all certificates to meet the standard specified.
      a. Click **Security > SSL certificate and key management > Manage FIPS > Convert Certificate**

         Note: If there are any certificates in the box labeled **Certificates** that can not be converted, then a certificate can not be converted using this option.
      b. Select the **Strict** radio button and choose which signatureAlgorithm to use when creating the new certificates from the pull-down box.
      c. Select the size of the certificate from the pull-down box labeled **New Certificate Key Size**. Note that Elliptical Curve signature algorithms require a specific size, so there is no need to provide a size.
      d. Click **Apply/Save**.

         The convertCertForSecurityStandard scripting task can also be used to convert all certificates to meet a specified standard.
    - Use the personal certificate panels to create new certificates and replace a certificate that does not meet the requirements by doing the following:
      a. Click **Security > SSL certificate and key management > Key stores and certificates**.
      b. Select a keystore from the collection panel.
      c. Select **Personal Certificate**.
         1) From the pull-down list on the Create button, select **Self-Signed Certificate**.
         2) Fill in an alias for the certificate. Select a signature algorithm for the certificate that is signed with SHA256, SHA384, or SHA512. Choose a size that is 2048 or greater. Note that Elliptical Curve signature algorithms require a specific size, so there is no need to specify a size.
         3) Click **Apply/Save**.

4) Go back to the Personal certificate collection panel and select the certificate that does not meet the standard. Click **Replace**.
5) On the Replace panel, select the certificate created that meets the standard from the pull-down list in the box labeled **Replace with**.
6) Select **Delete old certificate** after replacement, and **Delete old signer boxes**.
7) Click **Apply/Save**.

Note: To replace chained certificates, a root certificate must be created that meets the standard. Follow the previous navigation path to the root certificate in the defaultRootStore, then create a chained certificate with that new root certificate.

The createSelfSigneCertificate scripting task can also be used to create self-signed certificate. The replaceCertificate scripting task can also be used to replace the new certificate for the old one.

- Use the personal certificate panels to import certificates and to replace the certificate that does not meet the requirements. Some certificate come from external sources such as a Certificate Authority (CA).
  a. Click **Security > SSL certificate and key management > Key stores and certificates**.
  b. Select a keystore from the collection panel.
  c. Select **Personal Certificate**.
     1) Select **Import Certificate**.
     2) Fill in the information needed to access the certificate in an existing keystore file.
     3) Click **Apply/Save**.
     4) Go back to the Personal certificate collection panel and select the certificate that does not meet the standard. Click the **Replace** button.
     5) On the Replace panel, select the certificate created that meets the standard from the pull-down list in the box labeled **Replace with**. Select **Delete old certificate after replacement** and **Delete old signer boxes**.
     6) Click **Apply/Save**.

     The importCertificate scripting task can also be used to import a certificate. The replaceCertificate scripting task can also be used to replace the new certificate for the old one.

12. To enable strict SP800-131, click **Security > SSL certificate and key management > Manage FIPS**.
13. Click the **Enable SP800-131**.
14. Click the **Strict**.
15. Click **Apply/Save**.
16. Restart your servers and manually sync your nodes for the changes to take effect.
17. Configure the client `ssl.client.props` file for the transition mode SP800-131 standard. Once the server is configured for SP800-131 transition mode, the `ssl.client.props` file might need to modified so that the admin client can connect to the server.

    Edit the `ssl.client.props` file. Change the com.ibm.ssl.protocol property to match the protocol the server is using.
18. Configure the `client ssl.client.props` file for strict mode SP800-131 standard. Once the server is configured for SP800-131 strict mode, the `ssl.client.props` file must be modified so that the admin client is running in SP800-131 strict mode. It is not able to make a SSL connection to the server without the change.

    Edit the `ssl.client.props` file as follows:
    a. Modify the com.ibm.security.useFIPS to be set to`true`.
    b. Add the com.ibm.websphere.security.FIPSLevel=SP800-131 just below the useFips property.

c. Change the com.ibm.ssl.protocol property to TLSv1.2

What to do next

The browser used to access the administrative console or an application must use a protocol that is compatible with the server. If the server is running in a transition mode, the browser must be set to use the protocol that matches the server. The SP800-131 standard requires that the SSL connection use the TLSv1.2 protocol, so the browser must support TLSv1.2 and use it to access the administrative console.

Configuring WebSphere Application Server for SP800-131 standard strict mode

You can configure WebSphere Application Server to use the SP800-131 standard strict mode.

Before you begin

Read the “WebSphere Application Server security standards configurations” topic for more background information regarding security standards.

About this task

The National Institute of Standards and Technology (NIST) Special Publications (SP) 800-131 standard strengthens algorithms and increases the key lengths to improve security. The standard also provides for a transition period to move to the new standard. The transition period enables a user to run in a mixed environment of settings not supported under the standard along with those that are supported. The NIST SP800-131 standard requires that users be configured for strict enforcement of the standard by a specific timeframe. See The National Institute of Standards and Technology web site for more details.

WebSphere Application Server can be configured to run SP800-131 in a `transition` mode or a `strict` mode. For instructions on how to configure transition mode, read the topic “Transitioning WebSphere Application Server to the SP800-131 Security Standard”.

To run in strict mode, there are several changes necessary to the server configuration:

- Secure Sockets Layer (SSL) configuration must use the TLSv1.2 protocol.
- The com.ibm.jsse2.sp800-131 system property must be set to `strict` for the JSSE to run in a strict SP800-131 mode.
- Certificates used for SSL communication must have a minimum length of 2048, and for Elliptical Curve (EC) certificates they must have a minimum length of 244.
- Certificates must be signed with a signature algorithm of SHA256, SHA384, or SHA512.
- SP800-131 approved cipher suites must be used.

.

Procedure

1. Click **Security > SSL certificate and key management > Manage FIPS** To run in a strict SP800-131 mode, all of the certificates used for SSL on the server must be converted to certificates that comply with the SP800-131 requirements.
2. To convert certificates, under Related Items, click **Convert Certificates**.
3. Select the radio button marked **Strict**, and choose which signatureAlgorithm to use when creating the new certificates from the pull-down box.
4. Select the size of the certificate from the pull-down box labeled **New Certificate Key Size**.

   Note: If you choose an Elliptical Curve signature algorithm, they require specific sizes; you are not able to fill in a size. The correct size is used instead.

5. If no certificates are displayed in the box labeled **Certificates that can not be converted**, click **Apply/Save**.
6. If certificates are displayed in the box labeled **Certificates that can not be converted**, the server is unable to convert the certificate for you. You must replace these certificates with ones that meet SP800-131 requirements. Reasons why the server can not convert a certificate for you include:
   - The certificate was created by a Certificate Authority (CA)
   - The certificate is in a read only keystore

   Once certificates are converted to meet the SP800-131 specification, perform the following steps to enable SP800-131 strict mode.
7. Click **SSL certificate and key management > Manage FIPS**.
8. Enable the radio button labeled **Enable SP800-131**.
9. Enable the radio button labeled **Strict**.
10. Click **Apply/Save**.
11. Restart the servers and manually sync the nodes for the SP800-131 strict mode to take effect.

    When these changes are applied, and the server is restarted, all of the SSL configuration on the server are modified to use the TLSv1.2 protocol and the com.ibm.jsse2.sp800-131 system property is set to `strict`. The SSL configuration uses the appropriate SSL ciphers for the standard.

    There are several wsadmin tasks that can be used to enable strict SP800-131 using scripting
    - Check the status of certificates for the security standard by using the listCertStatusForSecurityStandard task.
    - Convert certificates for the security standard by using the convertCertForSecurityStandard task.
    - Enable the security standard by using the enableFips task.
    - To see the security standard setting, use the getFipsInfo task.
12. Once the server is configured for SP800-131 strict mode, the `ssl.client.props` file must be modified so that the admininstrative client is running in SP800-131 strict mode. They are not able to make a SSL connection to the server without the change.

    Edit the `ssl.client.props` file by doing the following:
    a. Modify com.ibm.security.useFIPS to be set to `true`.
    b. Add com.ibm.websphere.security.FIPSLevel=SP800-131 just below the useFips property.
    c. Change the com.ibm.ssl.protocol property to TLSv1.2.

What to do next

The SP800-131 standard strict mode requires that the SSL connection use the TLSv1.2 protocol. For a browser to access the administrative console or an application, the browser must support and first be configured to use the TLSv1.2 protocol.

Manually sync the nodes with syncNode, and start the node agents and servers. To use syncNode, you might need to update the `ssl.client.props` file to communicate with the deployment manager.

Implementing tokens for security attribute propagation

As part of an extensible architecture, WebSphere Application Server enables you to implement your own tokens in which to propagate security attributes.

About this task

The following topics are covered in this section:

Procedure

- Implementing a custom propagation token
- Implementing a custom authorization token
- Implementing a custom a single sign-on token
- Implementing a custom authentication token
- Propagating a custom Java serializable object

Implementing a custom propagation token for security attribute propagation

This topic explains how you might create your own propagation token implementation, which is set on the running thread and propagated downstream.

About this task

The default propagation token usually is sufficient for propagating attributes that are not user-specific. Consider writing your own implementation if you want to accomplish one of the following tasks:

- Isolate your attributes within your own implementation.
- Serialize the information using custom serialization. You must deserialize the bytes at the target and add that information back on the thread by plugging in a custom login module into the inbound system login configurations. This task also might include encryption and decryption.

To implement a custom propagation token, you must complete the following steps:

Procedure

1. Write a custom implementation of the PropagationToken interface. Many different methods are available for implementing the PropagationToken interface. However, make sure that the methods that are required by the PropagationToken interface and the token interface are fully implemented.

   After you implement this interface, you can place it in the *app_server_root*/classes directory. Alternatively, you can place the class in any private directory. However, make sure that the WebSphere Application Server class loader can locate the class and that it is granted the appropriate permissions. You can add the Java archive (JAR) file or directory that contains this class into the server.policy file so that it has the required permissions for the server code.

   Tip: All of the token types that are defined by the propagation framework have similar interfaces. The token types are marker interfaces that implement the com.ibm.wsspi.security.token.Token interface. This interface defines most of the methods. If you plan to implement more than one token type, consider creating an abstract class that implements the com.ibm.wsspi.security.token.Token interface. All of your token implementations, including the propagation token, might extend the abstract class and then most of the work is complete.

   To see an implementation of the propagation token, see “Example: com.ibm.wsspi.security.token.PropagationToken implementation” on page 889.
2. Add and receive the custom propagation token during WebSphere Application Server logins. This task is typically accomplished by adding a custom login module to the various application and system login configurations. You also can add the implementation from an application. However, to deserialize the information, you need to plug in a custom login module, which is discussed in “Propagating a custom Java serializable object for security attribute propagation” on page 916. The WSSecurityPropagationHelper class has APIs that are used to set a propagation token on the thread and to retrieve the token from the thread to make updates.

   The code sample in “Example: Custom propagation token login module” on page 893 shows how to determine if the login is an initial login or a propagation login. The difference between these login types is whether the WSTokenHolderCallback callback contains propagation data. If the callback does not contain propagation data, initialize a new custom propagation token implementation and set it on the

thread. If the callback contains propagation data, look for your specific custom propagation token TokenHolder instance, convert the byte array back into your custom PropagationToken object, and set it back on the thread. The code sample shows both instances.

You can add attributes any time your custom propagation token is added to the thread. If you add attributes between requests and the getUniqueId method changes, the Common Secure Interoperability Version 2 (CSIv2) client session is invalidated so that it can send the new information downstream. Adding attributes between requests can affect performance. In many cases, you want the downstream requests to receive the new propagation token information.

To add the custom propagation token to the thread, call the WSSecurityPropagationHelper.addPropagationToken method. This call requires the WebSphereRuntimePerMission "setPropagationToken" Java 2 Security permission.

3. Add your custom login module to WebSphere Application Server system login configurations that already contain the com.ibm.ws.security.server.lm.wsMapDefaultInboundLoginModule login module for receiving serialized versions of your custom propagation token You can also add this login module to any of the application logins where you might want to generate your custom propagation token on the thread during the login. Alternatively, you can generate the custom PropagationToken implementation from within your application. However, to deserialize it, you need to add the implementation to the system login modules.

Results

After completing these steps, you have implemented a custom PropagationToken.

Example: com.ibm.wsspi.security.token.PropagationToken implementation

Use this file to see an example of a propagation token implementation. The following sample code does not extend an abstract class, but implements the com.ibm.wsspi.security.token.PropagationToken interface directly. You can implement the interface directly, but it might cause you to write duplicate code. However, you might choose to implement the interface directly if considerable differences exist between how you handle the various token implementations.

For information on how to implement a custom propagation token, see “Implementing a custom propagation token for security attribute propagation” on page 888.

```
package com.ibm.websphere.security.token;

import com.ibm.websphere.security.WSSecurityException;
import com.ibm.websphere.security.auth.WSLoginFailedException;
import com.ibm.wsspi.security.token.*;
import com.ibm.websphere.security.WebSphereRuntimePermission;
import java.io.ByteArrayOutputStream;
import java.io.ByteArrayInputStream;
import java.io.DataOutputStream;
import java.io.DataInputStream;
import java.io.ObjectOutputStream;
import java.io.ObjectInputStream;
import java.io.OutputStream;
import java.io.InputStream;
import java.util.ArrayList;

public class CustomPropagationTokenImpl implements com.ibm.wsspi.security.
   token.PropagationToken
{
 private java.util.Hashtable hashtable = new java.util.Hashtable();
 private byte[] tokenBytes = null;
   // 2 hours in millis, by default
  private static long expire_period_in_millis = 2*60*60*1000;
 private long counter = 0;

/**
 * The constructor that is used to create initial PropagationToken instance
 */

 public CustomAbstractTokenImpl ()
 {
  // set the token version
  addAttribute("version", "1");
  // set the token expiration
  addAttribute("expiration", new Long(System.currentTimeMillis() +
        expire_period_in_millis).toString());
 }
```

```
/**
 * The constructor that is used to deserialize the token bytes received
 * during a propagation login.
 */
 public CustomAbstractTokenImpl (byte[] token_bytes)
 {
  try
  {
       hashtable = (java.util.Hashtable) com.ibm.wsspi.security.token.
         WSOpaqueTokenHelper.deserialize(token_bytes);
  }
  catch (Exception e)
  {
   e.printStackTrace();
  }
 }

/**
 * Validates the token including expiration, signature, and so on.
 * @return boolean
 */

 public boolean isValid ()
 {
  long expiration = getExpiration();

  // if you set the expiration to 0, it does not expire
  if (expiration != 0)
  {
   // return if this token is still valid
   long current_time = System.currentTimeMillis();

   boolean valid = ((current_time < expiration) ? true : false);
   System.out.println("isValid: returning " + valid);
   return valid;
  }
  else
  {
   System.out.println("isValid: returning true by default");
   return true;
  }
 }

/**
 * Gets the expiration as a long type.
 * @return long
 */
 public long getExpiration()
 {
  // get the expiration value from the hashtable
  String[] expiration = getAttributes("expiration");

  if (expiration != null && expiration[0] != null)
  {
   // expiration is the first element (should only be one)
   System.out.println("getExpiration: returning " + expiration[0]);
   return new Long(expiration[0]).longValue();
  }

  System.out.println("getExpiration: returning 0");
  return 0;
 }

/**
 * Returns if this token should be forwarded/propagated downstream.
 * @return boolean
 */
 public boolean isForwardable()
 {
     // You can choose whether your token gets propagated. In some cases
     // you might want the token to be local only.
  return true;
 }

/**
 * Gets the principal that this token belongs to. If this token is an
 * authorization token, this principal string must match the authentication
 * token principal string or the message is rejected.
 * @return String
 */
 public String getPrincipal()
 {
  // It is not necessary for the PropagationToken to return a principal,
     // because it is not user-centric.
  return "";
 }

/**
 * Returns the unique identifier of the token based upon information that
```

```
 * the provider considers makes it a unique token. This identifier is used
 * for caching purposes and might be used in combination with other token
 * unique IDs that are part of the same Subject.
 *
 * This method should return null if you want the accessID of the user to
 * represent its uniqueness. This is the typical scenario.
 *
 * @return String
 */
 public String getUniqueID()
 {
     // If you want to propagate the changes to this token, change the
     // value that this unique ID returns whenever the token is changed.
     // Otherwise, CSIv2 uses an existing session when everything else is
     // the same. This getUniqueID is checked by CSIv2 to determine the
     // session lookup.
  return counter;
 }

/**
 * Gets the bytes to be sent across the wire. The information in the byte[]
 * needs to be enough to recreate the Token object at the target server.
 * @return byte[]
 */
 public byte[] getBytes ()
 {
  if (hashtable != null)
  {
   try
   {
     // Do this if the object is set to read-only during login commit
     // because this guarantees that no new data is set.
    if (isReadOnly() && tokenBytes == null)
     tokenBytes = com.ibm.wsspi.security.token.WSOpaqueTokenHelper.
               serialize(hashtable);

     // You can deserialize this in the downstream login module using
         //  WSOpaqueTokenHelper.deserialize()
    return tokenBytes;
   }
   catch (Exception e)
   {
    e.printStackTrace();
    return null;
   }
  }

  System.out.println("getBytes: returning null");
  return null;
 }

/**
 * Gets the name of the token, which is used to identify the byte[] in the
 * protocol message.
 * @return String
 */
 public String getName()
 {
  return this.getClass().getName();
 }

/**
 * Gets the version of the token as a short type. This code also is used
 * to identify the byte[] in the protocol message.
 * @return short
 */
 public short getVersion()
 {
  String[] version = getAttributes("version");

  if (version != null && version[0] != null)
   return new Short(version[0]).shortValue();

  System.out.println("getVersion: returning default of 1");
  return 1;
    }

/**
 * When called, the token becomes irreversibly read-only. The implementation
 * needs to ensure that any setter methods check that this read-only flag has
 * been set.
 */
 public void setReadOnly()
 {
  addAttribute("readonly", "true");
 }

/**
 * Called internally to see if the token is readonly
 */
```

```
 private boolean isReadOnly()
 {
  String[] readonly = getAttributes("readonly");

  if (readonly != null && readonly[0] != null)
   return new Boolean(readonly[0]).booleanValue();

  System.out.println("isReadOnly: returning default of false");
  return false;
 }

/**
 * Gets the attribute value based on the named value.
 * @param String key
 * @return String[]
 */
 public String[] getAttributes(String key)
 {
  ArrayList array = (ArrayList) hashtable.get(key);

  if (array != null && array.size() > 0)
  {
   return (String[]) array.toArray(new String[0]);
  }

  return null;
 }

/**
 * Sets the attribute name and value pair. Returns the previous values set
 * for the key, or returns null if the value is not previously set.
 * @param String key
 * @param String value
 * @returns String[];
 */
 public String[] addAttribute(String key, String value)
 {
  // Gets the current value for the key
  ArrayList array = (ArrayList) hashtable.get(key);

  if (!isReadOnly())
  {
   // Increments the counter to change the uniqueID
   counter++;

   // Copies the ArrayList to a String[] as it currently exists
   String[] old_array = null;
   if (array != null && array.size() > 0)
    old_array = (String[]) array.toArray(new String[0]);

   // Allocates a new ArrayList if one was not found
   if (array == null)
    array = new ArrayList();

   // Adds the String to the current array list
   array.add(value);

   // Adds the current ArrayList to the Hashtable
   hashtable.put(key, array);

   // Returns the old array
   return old_array;
  }

  return (String[]) array.toArray(new String[0]);
 }


/**
 * Gets the list of all of the attribute names present in the token.
 * @return java.util.Enumeration
 */
 public java.util.Enumeration getAttributeNames()
 {
  return hashtable.keys();
 }

/**
 * Returns a deep clone of this token. This is typically used by the session
 * logic of the CSIv2 server to create a copy of the token as it exists in the
 * session.
 * @return Object
 */
 public Object clone()
 {
  com.ibm.websphere.security.token.CustomPropagationTokenImpl deep_clone =
   new com.ibm.websphere.security.token.CustomPropagationTokenImpl();

  java.util.Enumeration keys = getAttributeNames();
```

```
  while (keys.hasMoreElements())
  {
   String key = (String) keys.nextElement();

   String[] list = (String[]) getAttributes(key);

   for (int i=0; i<list.length; i++)
    deep_clone.addAttribute(key, list[i]);
  }

      return deep_clone;
 }
}
```

Example: Custom propagation token login module

This example shows how to determine if the login is an initial login or a propagation login.

```
public customLoginModule()
{
 public void initialize(Subject subject, CallbackHandler callbackHandler,
        Map sharedState, Map options)
 {
  // (For more information on what to do during initialization, see
     // Developing custom login modules for a system login configuration for JAAS.)
 }

 public boolean login() throws LoginException
 {
  // (For more information on what to do during login, see
     // Developing custom login modules for a system login configuration for JAAS.)

  // Handles the WSTokenHolderCallback to see if this is an initial
     // or propagation login.
  Callback callbacks[] = new Callback[1];
  callbacks[0] = new WSTokenHolderCallback("Authz Token List: ");

  try
  {
   callbackHandler.handle(callbacks);
  }
  catch (Exception e)
  {
   // handle exception
  }

  // Receives the ArrayList of TokenHolder objects (the serialized tokens)
  List authzTokenList = ((WSTokenHolderCallback) callbacks[0]).getTokenHolderList();

  if (authzTokenList != null)
  {
   // Iterates through the list looking for your custom token
   for (int i=0; i<authzTokenList.size(); i++)
   {
    TokenHolder tokenHolder = (TokenHolder)authzTokenList.get(i);

    // Looks for the name and version of your custom PropagationToken implementation
    if (tokenHolder.getName().equals("
              com.ibm.websphere.security.token.CustomPropagationTokenImpl") &&
        tokenHolder.getVersion() == 1)
    {
     // Passes the bytes into your custom PropagationToken constructor
            //  to deserialize
     customPropToken = new
      com.ibm.websphere.security.token.CustomPropagationTokenImpl(tokenHolder.
                 getBytes());

    }
   }
  }
  else // This is not a propagation login. Create a new instance of
          // your PropagationToken implementation
  {
   // Adds a new custom propagation token. This is an initial login
   customPropToken = new com.ibm.websphere.security.token.CustomPropagationTokenImpl();

   // Adds any initial attributes
   if (customPropToken != null)
   {
    customPropToken.addAttribute("key1", "value1");
    customPropToken.addAttribute("key1", "value2");
    customPropToken.addAttribute("key2", "value1");
    customPropToken.addAttribute("key3", "something different");
   }
  }

  // Note: You can add the token to the thread during commit in case
     // something happens during the login.
 }
```

```
public boolean commit() throws LoginException
{
 // For more information on what to do during commit, see
    // Developing custom login modules for a system login configuration for JAAS.
 if (customPropToken != null)
 {
  // Sets the propagation token on the thread
  try
  {

   System.out.println(tc, "*** ADDED MY CUSTOM PROPAGATION TOKEN TO THE THREAD ***");
   // Prints out the values in the deserialized propagation token
   java.util.Enumeration keys = customPropToken.getAttributeNames();
   while (keys.hasMoreElements())
   {
    String key = (String) keys.nextElement();
    String[] list = (String[]) customPropToken.getAttributes(key);
    for (int k=0; k<list.length; k++)
    System.out.println("Key/Value: " + key + "/" + list[k]);
   }

   // This sets it on the thread using getName() + getVersion() as the key
   com.ibm.wsspi.security.token.WSSecurityPropagationHelper.addPropagationToken(
             customPropToken);
  }
  catch (Exception e)
  {
   // Handles exception
  }


  // Now you can verify that you have set it properly by trying to get
      // it back from the thread and print the values.
  try
  {
   // This gets the PropagationToken from the thread using getName()
         // and getVersion() parameters.
   com.ibm.wsspi.security.token.PropagationToken tempPropagationToken =
    com.ibm.wsspi.security.token.WSSecurityPropagationHelper.getPropagationToken
     ("com.ibm.websphere.security.token.CustomPropagationTokenImpl", 1);

   if  (tempPropagationToken != null)
   {
    System.out.println(tc, "*** RECEIVED MY CUSTOM PROPAGATION
              TOKEN FROM THE THREAD ***");
    // Prints out the values in the deserialized propagation token
    java.util.Enumeration keys = tempPropagationToken.getAttributeNames();
    while (keys.hasMoreElements())
    {
     String key = (String) keys.nextElement();
     String[] list = (String[]) tempPropagationToken.getAttributes(key);
     for (int k=0; k<list.length; k++)
     System.out.println("Key/Value: " + key + "/" + list[k]);
    }
   }
  }
  catch (Exception e)
  {
   // Handles exception
  }
 }
}

// Defines your login module variables
com.ibm.wsspi.security.token.PropagationToken customPropToken = null;

}
```

Implementing a custom authorization token for security attribute propagation

This task explains how you might create your own AuthorizationToken implementation, which is set in the login Subject and propagated downstream.

About this task

The default AuthorizationToken usually is sufficient for propagating attributes that are user-specific. Consider writing your own implementation if you want to accomplish one of the following tasks:

- Isolate your attributes within your own implementation.
- Serialize the information using custom serialization. You must deserialize the bytes at the target and add that information back on the thread. This task also might include encryption and decryption.

- Affect the overall uniqueness of the Subject using the getUniqueID() application programming interface (API).

To implement a custom authorization token, you must complete the following steps:

Procedure

1. Write a custom implementation of the AuthorizationToken interface. There are many different methods for implementing the AuthorizationToken interface. However, make sure that the methods required by the AuthorizationToken interface and the token interface are fully implemented.

   After you implement this interface, you can place it in the *app_server_root*/classes directory. Alternatively, you can place the class in any private directory. However, make sure that the WebSphere Application Server class loader can locate the class and that it is granted the appropriate permissions. You can add the Java archive (JAR) file or directory that contains this class into the server.policy file so that it has the necessary permissions that are needed by the server code.

   Tip: All of the token types defined by the propagation framework have similar interfaces. Basically, the token types are marker interfaces that implement the com.ibm.wsspi.security.token.Token interface. This interface defines most of the methods. If you plan to implement more than one token type, consider creating an abstract class that implements the com.ibm.wsspi.security.token.Token interface. All of your token implementations, including the AuthorizationToken, might extend the abstract class and then most of the work is completed.

   To see an implementation of AuthorizationToken, see "Example: com.ibm.wsspi.security.token.AuthorizationToken implementation" on page 896
2. Add and receive the custom AuthorizationToken during WebSphere Application Server logins. This task is typically accomplished by adding a custom login module to the various application and system login configurations. However, in order to deserialize the information, you must plug in a custom login module, which is discussed in "Propagating a custom Java serializable object for security attribute propagation" on page 916. After the object is instantiated in the login module, you can add the object to the Subject during the commit() method.

   If you only want to add information to the Subject to get propagated, see "Propagating a custom Java serializable object for security attribute propagation" on page 916. If you want to ensure that the information is propagated, want to do you own custom serialization, or want to specify the uniqueness for Subject caching purposes, then consider writing your own AuthorizationToken implementation.

   The code sample in "Example: custom AuthorizationToken login module" on page 899 shows how to determine if the login is an initial login or a propagation login. The difference between these login types is whether the WSTokenHolderCallback contains propagation data. If the callback does not contain propagation data, initialize a new custom AuthorizationToken implementation and set it into the Subject. If the callback contains propagation data, look for your specific custom AuthorizationToken TokenHolder instance, convert the byte[] back into your custom AuthorizationToken object, and set it back into the Subject. The code sample shows both instances.

   You can make your AuthorizationToken read-only in the commit phase of the login module. If you do not make the token read-only, then attributes can be added within your applications.
3. Add your custom login module to WebSphere Application Server system login configurations that already contain the com.ibm.ws.security.server.lm.wsMapDefaultInboundLoginModule for receiving serialized versions of your custom authorization token.

   Because this login module relies on information in the sharedState added by the com.ibm.ws.security.server.lm.wsMapDefaultInboundLoginModule, add this login module after com.ibm.ws.security.server.lm.wsMapDefaultInboundLoginModule. For information on how to add your custom login module to the existing login configurations, see Developing custom login modules for a system login configuration for JAAS.

Results

After completing these steps, you have implemented a custom AuthorizationToken.

Example: com.ibm.wsspi.security.token.AuthorizationToken implementation

Use this file to see an example of a AuthorizationToken implementation. The following sample code does not extend an abstract class, but rather implements the com.ibm.wsspi.security.token.AuthorizationToken interface directly. You can implement the interface directly, but it might cause you to write duplicate code. However, you might choose to implement the interface directly if there are considerable differences between how you handle the various token implementations.

For information on how to implement a custom AuthorizationToken, see “Implementing a custom authorization token for security attribute propagation” on page 894.

```
package com.ibm.websphere.security.token;

import com.ibm.websphere.security.WSSecurityException;
import com.ibm.websphere.security.auth.WSLoginFailedException;
import com.ibm.wsspi.security.token.*;
import com.ibm.websphere.security.WebSphereRuntimePermission;
import java.io.ByteArrayOutputStream;
import java.io.ByteArrayInputStream;
import java.io.DataOutputStream;
import java.io.DataInputStream;
import java.io.ObjectOutputStream;
import java.io.ObjectInputStream;
import java.io.OutputStream;
import java.io.InputStream;
import java.util.ArrayList;

public class CustomAuthorizationTokenImpl implements com.ibm.wsspi.security.
   token.AuthorizationToken
{
 private java.util.Hashtable hashtable = new java.util.Hashtable();
 private byte[] tokenBytes = null;
 private static long expire_period_in_millis = 2*60*60*1000;
  // 2 hours in millis, by default

/**
 * Constructor used to create initial AuthorizationToken instance
 */

 public CustomAuthorizationTokenImpl (String principal)
 {
  // Sets the principal in the token
  addAttribute("principal", principal);
  // Sets the token version
  addAttribute("version", "1");
  // Sets the token expiration
  addAttribute("expiration", new Long(System.currentTimeMillis() +
        expire_period_in_millis).toString());
 }

/**
 * Constructor used to deserialize the token bytes received during a
 * propagation login.
 */
 public CustomAuthorizationTokenImpl (byte[] token_bytes)
 {
  try
  {
   hashtable = (java.util.Hashtable) com.ibm.wsspi.security.token.
           WSOpaqueTokenHelper.deserialize(token_bytes);
  }
  catch (Exception e)
  {
   e.printStackTrace();
  }
 }

/**
 * Validates the token including expiration, signature, and so on.
 * @return boolean
 */

 public boolean isValid ()
 {
  long expiration = getExpiration();

  // if you set the expiration to 0, it does not expire
  if (expiration != 0)
  {
   // return if this token is still valid
   long current_time = System.currentTimeMillis();

   boolean valid = ((current_time < expiration) ? true : false);
   System.out.println("isValid: returning " + valid);
   return valid;
  }
```

```
  else
  {
   System.out.println("isValid: returning true by default");
   return true;
  }
 }

/**
 * Gets the expiration as a long.
 * @return long
 */
 public long getExpiration()
 {
  // Gets the expiration value from the hashtable
  String[] expiration = getAttributes("expiration");

  if (expiration != null && expiration[0] != null)
  {
   // The expiration is the first element. There should be only one expiration.
   System.out.println("getExpiration: returning " + expiration[0]);
   return new Long(expiration[0]).longValue();
  }

  System.out.println("getExpiration: returning 0");
  return 0;
 }

/**
 * Returns if this token should be forwarded/propagated downstream.
 * @return boolean
 */
 public boolean isForwardable()
 {
  // You can choose whether your token gets propagated. In some cases,
     // you might want it to be local only.
  return true;
 }

/**
 * Gets the principal that this Token belongs to. If this is an authorization token,
 * this principal string must match the authentication token principal string or the
 * message will be rejected.
 * @return String
 */
 public String getPrincipal()
 {
  // this might be any combination of attributes
  String[] principal = getAttributes("principal");

  if (principal != null && principal[0] != null)
  {
   return principal[0];
  }

  System.out.println("getExpiration: returning null");
  return null;
 }

/**
 * Returns a unique identifier of the token based upon the information that provider
 * considers makes this a unique token. This will be used for caching purposes
 * and might be used in combination with other token unique IDs that are part of
 * the same Subject.
 *
 * This method should return null if you want the accessID of the user to represent
 * uniqueness.  This is the typical scenario.
 *
 * @return String
 */
 public String getUniqueID()
 {
  // if you don't want to affect the cache lookup, just return NULL here.
  // return null;

  String cacheKeyForThisToken = "dynamic attributes";

  // if you do want to affect the cache lookup, return a string of
     // attributes that you want factored into the lookup.
  return cacheKeyForThisToken;
 }

/**
 * Gets the bytes to be sent across the wire. The information in the byte[]
 * needs to be enough to recreate the Token object at the target server.
 * @return byte[]
 */
 public byte[] getBytes ()
 {
  if (hashtable != null)
  {
```

```
   try
   {
    // Do this if the object is set to read-only during login commit,
    // because this makes sure that no new data gets set.
    if (isReadOnly() && tokenBytes == null)
     tokenBytes = com.ibm.wsspi.security.token.WSOpaqueTokenHelper.
                 serialize(hashtable);

    // You can deserialize this in the downstream login module using
          // WSOpaqueTokenHelper.deserialize()
    return tokenBytes;
   }
   catch (Exception e)
   {
    e.printStackTrace();
    return null;
   }
  }

  System.out.println("getBytes: returning null");
  return null;
 }

/**
 * Gets the name of the token used to identify the byte[] in the protocol message.
 * @return String
 */
 public String getName()
 {
  return this.getClass().getName();
 }

/**
 * Gets the version of the token as an short. This also is used to identify the
 * byte[] in the protocol message.
 * @return short
 */
 public short getVersion()
 {
  String[] version = getAttributes("version");

  if (version != null && version[0] != null)
   return new Short(version[0]).shortValue();

  System.out.println("getVersion: returning default of 1");
  return 1;
    }

/**
 * When called, the token becomes irreversibly read-only. The implementation
 * needs to ensure that any setter methods check that this flag has been set.
 */
 public void setReadOnly()
 {
  addAttribute("readonly", "true");
 }

/**
 * Called internally to see if the token is read-only
 */
 private boolean isReadOnly()
 {
  String[] readonly = getAttributes("readonly");

  if (readonly != null && readonly[0] != null)
   return new Boolean(readonly[0]).booleanValue();

  System.out.println("isReadOnly: returning default of false");
  return false;
 }

/**
 * Gets the attribute value based on the named value.
 * @param String key
 * @return String[]
 */
 public String[] getAttributes(String key)
 {
  ArrayList array = (ArrayList) hashtable.get(key);

  if (array != null && array.size() > 0)
  {
   return (String[]) array.toArray(new String[0]);
  }

  return null;
 }

/**
 * Sets the attribute name and value pair. Returns the previous values set for key,
```

```
 * or null if not previously set.
 * @param String key
 * @param String value
 * @returns String[];
 */
 public String[] addAttribute(String key, String value)
 {
  // Gets the current value for the key
  ArrayList array = (ArrayList) hashtable.get(key);

  if (!isReadOnly())
  {
   // Copies the ArrayList to a String[] as it currently exists
   String[] old_array = null;
   if (array != null && array.size() > 0)
    old_array = (String[]) array.toArray(new String[0]);

   // Allocates a new ArrayList if one was not found
   if (array == null)
    array = new ArrayList();

   // Adds the String to the current array list
   array.add(value);

   // Adds the current ArrayList to the Hashtable
   hashtable.put(key, array);

   // Returns the old array
   return old_array;
  }

  return (String[]) array.toArray(new String[0]);
 }


/**
 * Gets the list of all attribute names present in the token.
 * @return java.util.Enumeration
 */
 public java.util.Enumeration getAttributeNames()
 {
  return hashtable.keys();
 }

/**
 * Returns a deep copying of this token, if necessary.
 * @return Object
 */
 public Object clone()
 {
  com.ibm.websphere.security.token.CustomAuthorizationTokenImpl deep_clone =
   new com.ibm.websphere.security.token.CustomAuthorizationTokenImpl();

  java.util.Enumeration keys = getAttributeNames();

  while (keys.hasMoreElements())
  {
   String key = (String) keys.nextElement();

   String[] list = (String[]) getAttributes(key);

   for (int i=0; i<list.length; i++)
    deep_clone.addAttribute(key, list[i]);
  }

      return deep_clone;
 }
}
```

Example: custom AuthorizationToken login module

This file shows how to determine if the login is an initial login or a propagation login.

For information on what to do during initialization, login and commit, see Developing custom login modules for a system login configuration for JAAS.

```
public customLoginModule()
{
 public void initialize(Subject subject, CallbackHandler callbackHandler,
     Map sharedState, Map options)
 {
  _sharedState = sharedState;
 }

 public boolean login() throws LoginException
 {
  // Handles the WSTokenHolderCallback to see if this is an initial or
     // propagation login.
```

```
 Callback callbacks[] = new Callback[1];
 callbacks[0] = new WSTokenHolderCallback("Authz Token List: ");

 try
 {
  callbackHandler.handle(callbacks);
 }
 catch (Exception e)
 {
  // Handles exception
 }

 // Receives the ArrayList of TokenHolder objects (the serialized tokens)
 List authzTokenList = ((WSTokenHolderCallback) callbacks[0]).getTokenHolderList();

 if (authzTokenList != null)
 {
  // Iterates through the list looking for your custom token
  for (int i=0; i
  for (int i=0; i<authzTokenList.size(); i++)
  {
   TokenHolder tokenHolder = (TokenHolder)authzTokenList.get(i);

         // Looks for the name and version of your custom AuthorizationToken
         // implementation
   if (tokenHolder.getName().equals("com.ibm.websphere.security.token.
           CustomAuthorizationTokenImpl") &&
       tokenHolder.getVersion() == 1)
   {
          // Passes the bytes into your custom AuthorizationToken constructor
          // to deserialize
    customAuthzToken = new
     com.ibm.websphere.security.token.CustomAuthorizationTokenImpl(
                 tokenHolder.getBytes());

   }
  }
 }
 else
    // This is not a propagation login. Create a new instance of your
    // AuthorizationToken implementation
 {
      // Gets the prinicpal from the default AuthenticationToken. This must match
      // all tokens.
  defaultAuthToken = (com.ibm.wsspi.security.token.AuthenticationToken)
   sharedState.get(com.ibm.wsspi.security.auth.callback.Constants.WSAUTHTOKEN_KEY);
  String principal = defaultAuthToken.getPrincipal();

      // Adds a new custom authorization token. This is an initial login. Pass the
      // principal into the constructor
  customAuthzToken = new com.ibm.websphere.security.token.
          CustomAuthorizationTokenImpl(principal);

  // Adds any initial attributes
  if (customAuthzToken != null)
  {
   customAuthzToken.addAttribute("key1", "value1");
   customAuthzToken.addAttribute("key1", "value2");
   customAuthzToken.addAttribute("key2", "value1");
   customAuthzToken.addAttribute("key3", "something different");
  }
 }

    // Note: You can add the token to the Subject during commit in case something
    // happens during the login.
}

public boolean commit() throws LoginException
{
 if (customAut  // (hzToken != null)
 {
  // sSets the customAuthzToken token into the Subject
  try
  {
   public final AuthorizationToken customAuthzTokenPriv = customAuthzToken;
         // Do this in a doPrivileged code block so that application code does not
         // need to add additional permissions
   java.security.AccessController.doPrivileged(new java.security.PrivilegedAction()
   {
    public Object run()
    {
     try
     {
                // Adds the custom authorization token if it is not null
                //  and not already in the Subject
                             if ((customAuthzTokenPriv != null) &&
        (!subject.getPrivateCredentials().contains(customAuthzTokenPriv)))
      {
       subject.getPrivateCredentials().add(customAuthzTokenPriv);
      }
```

```
      }
      catch (Exception e)
      {
       throw new WSLoginFailedException (e.getMessage(), e);
      }

      return null;
     }
    });
   }
   catch (Exception e)
   {
    throw new WSLoginFailedException (e.getMessage(), e);
   }
  }
 }

 // Defines your login module variables
 com.ibm.wsspi.security.token.AuthorizationToken customAuthzToken = null;
 com.ibm.wsspi.security.token.AuthenticationToken defaultAuthToken = null;
 java.util.Map _sharedState = null;
}
```

Implementing a custom single sign-on token for security attribute propagation

You can create your own single sign-on token implementation. The single sign-on token implementation is set in the login Subject and added to the HTTP response as an HTTP cookie.

About this task

The cookie name is the concatenation of the SingleSignonToken.getName application programming interface (API) and the SingleSignonToken.getVersion API. There is no delimiter. When you add a single sign-on token to the Subject, it also gets propagated horizontally and downstream in case the Subject is used for other web requests. You must deserialize your custom single sign-on token when you receive it from a propagation login. Consider writing your own implementation if you want to accomplish one of the following tasks:

- Isolate your attributes within your own implementation.
- Serialize the information using custom serialization. Encrypt the information because it is out to the HTTP response and is available on the Internet. You must deserialize or decrypt the bytes at the target and add that information back into the Subject.
- Affect the overall uniqueness of the Subject using the getUniqueID API.

To implement a custom single sign-on token, complete the following steps:

Procedure

1. Write a custom implementation of the SingleSignonToken interface.

   Many different methods are available for implementing the SingleSignonToken interface. However, make sure the methods that are required by the SingleSignonToken interface and the token interface are fully implemented.

   After you implement this interface, you can place it in the *app_server_root*/classes directory. Alternatively, you can place the class in any private directory. However, make sure that the WebSphere Application Server class loader can locate the class and that it is granted the appropriate permissions. You can add the Java archive (JAR) file or directory that contains this class into the server.policy file so that it has the required permissions for the server code.

   Tip: All of the token types that are defined by the propagation framework have similar interfaces. Basically, the token types are marker interfaces that implement the com.ibm.wsspi.security.token.Token interface. This interface defines most of the methods. If you plan to implement more than one token type, consider creating an abstract class that implements the com.ibm.wsspi.security.token.Token interface. All of your token implementations, including the single sign-on token, might extend the abstract class and then most of the work is complete.

To see an implementation of the single sign-on token, see “Example: A com.ibm.wsspi.security.token.SingleSignonToken implementation”

2. Add and receive the custom single sign-on token during WebSphere Application Server logins. This task is typically accomplished by adding a custom login module to the various application and system login configurations. However, to deserialize the information, you need to plug in a custom login module, which is discussed in a subsequent step. After the object is instantiated in the login module, you can add it to the Subject during the commit method.

   The code sample in “Example: A custom single sign-on token login module” on page 906, shows how to determine if the login is an initial login or a propagation login. The difference is whether the WSTokenHolderCallback callback contains propagation data. If the callback does not contain propagation data, initialize a new custom single sign-on token implementation and set it into the Subject. Also, look for the HTTP cookie from the HTTP request if the HTTP request object is available in the callback. You can get your custom single sign-on token both from a horizontal propagation login and from the HTTP request. However, it is recommended that you make the token available in both places because then the information arrives at any front-end application server, even if that server does not support propagation.

   You can make your single sign-on token read-only in the commit phase of the login module. If you make the token read-only, additional attributes cannot be added within your applications.

   Restriction:

   - HTTP cookies have a size limitation. Size restrictions should be included in the documentation for your specific browser.
   - The WebSphere Application Server runtime does not handle cookies that it does not generate, so this cookie is not used by the runtime.
   - The SingleSignonToken object, when in the Subject, does affect the cache lookup of the Subject if you return something in the getUniqueID method.

3. Get the HTTP cookie from the HTTP request object during login or from an application. The sample code that is found in “Example: An HTTP cookie retrieval” on page 908 shows how you can retrieve the HTTP cookie from the HTTP request, decode the cookie so that it is back to your original bytes, and create your custom SingleSignonToken object from the bytes.
4. Add your custom login module to WebSphere Application Server system login configurations that already contain the com.ibm.ws.security.server.lm.wsMapDefaultInboundLoginModule for receiving serialized versions of your custom propagation token. Because this login module relies on information in the sharedState state that is added by the com.ibm.ws.security.server.lm.wsMapDefaultInboundLoginModule login module, add this login module after the com.ibm.ws.security.server.lm.wsMapDefaultInboundLoginModule login module.

   For information on adding your custom login module into the existing login configurations, see Developing custom login modules for a system login configuration for JAAS.

Results

After completing these steps, you have implemented a custom single sign-on token.

Example: A com.ibm.wsspi.security.token.SingleSignonToken implementation

Use this file to see an example of a single sign-on implementation. The following sample code does not extend an abstract class, but implements the com.ibm.wsspi.security.token.SingleSignonToken interface directly. You can implement the interface directly, but it might cause you to write duplicate code. However, you might choose to implement the interface directly if considerable differences exist between how you handle the various token implementations.

For information on how to implement a custom single sign-on token, see “Implementing a custom single sign-on token for security attribute propagation” on page 901.

```
package com.ibm.websphere.security.token;

import com.ibm.websphere.security.WSSecurityException;
import com.ibm.websphere.security.auth.WSLoginFailedException;
import com.ibm.wsspi.security.token.*;
import com.ibm.websphere.security.WebSphereRuntimePermission;
import java.io.ByteArrayOutputStream;
import java.io.ByteArrayInputStream;
import java.io.DataOutputStream;
import java.io.DataInputStream;
import java.io.ObjectOutputStream;
import java.io.ObjectInputStream;
import java.io.OutputStream;
import java.io.InputStream;
import java.util.ArrayList;

public class CustomSingleSignonTokenImpl implements com.ibm.wsspi.security.
   token.SingleSignonToken
{
 private java.util.Hashtable hashtable = new java.util.Hashtable();
 private byte[] tokenBytes = null;
   // 2 hours in millis, by default
 private static long expire_period_in_millis = 2*60*60*1000;

/**
 * Constructor used to create initial SingleSignonToken instance
 */

 public CustomSingleSignonTokenImpl (String principal)
 {
  // set the principal in the token
  addAttribute("principal", principal);
  // set the token version
  addAttribute("version", "1");
  // set the token expiration
  addAttribute("expiration", new Long(System.currentTimeMillis() +
        expire_period_in_millis).toString());
 }

/**
 * Constructor used to deserialize the token bytes received during a propagation login.
 */
 public CustomSingleSignonTokenImpl (byte[] token_bytes)
 {
  try
  {
   // you should implement a decryption algorithm to decrypt the cookie bytes
   hashtable = (java.util.Hashtable) some_decryption_algorithm (token_bytes);
  }
  catch (Exception e)
  {
   e.printStackTrace();
  }
 }

/**
 * Validates the token including expiration, signature, and so on.
 * @return boolean
 */

 public boolean isValid ()
 {
  long expiration = getExpiration();

  // if you set the expiration to 0, it does not expire
  if (expiration != 0)
  {
   // return if this token is still valid
   long current_time = System.currentTimeMillis();

   boolean valid = ((current_time < expiration) ? true : false);
   System.out.println("isValid: returning " + valid);
   return valid;
  }
  else
  {
   System.out.println("isValid: returning true by default");
   return true;
  }
 }

/**
 * Gets the expiration as a long.
 * @return long
 */
 public long getExpiration()
 {
  // get the expiration value from the hashtable
  String[] expiration = getAttributes("expiration");
```

```
  if (expiration != null && expiration[0] != null)
  {
   // expiration will always be the first element (should only be one)
   System.out.println("getExpiration: returning " + expiration[0]);
   return new Long(expiration[0]).longValue();
  }

  System.out.println("getExpiration: returning 0");
  return 0;
 }

/**
 * Returns if this token should be forwarded/propagated downstream.
 * @return boolean
 */
 public boolean isForwardable()
 {
  // You can choose whether your token gets propagated or not, in some cases
     // you might want it to be local only.
  return true;
 }

/**
 * Gets the principal that this Token belongs to. If this is an authorization token,
 * this principal string must match the authentication token principal string or the
 * message will be rejected.
 * @return String
 */
 public String getPrincipal()
 {
  // this could be any combination of attributes
  String[] principal = getAttributes("principal");

  if (principal != null && principal[0] != null)
  {
   return principal[0];
  }

  System.out.println("getExpiration: returning null");
  return null;
 }

/**
 * Returns a unique identifier of the token based upon information the provider
 * considers makes this a unique token.  This will be used for caching purposes
 * and may be used in combination with other token unique IDs that are part of
 * the same Subject.
 *
 * This method should return null if you want the access ID of the user to represent
 * uniqueness.  This is the typical scenario.
 *
 * @return String
 */
 public String getUniqueID()
 {
  // this could be any combination of attributes
  return getPrincipal();
 }

/**
 * Gets the bytes to be sent across the wire.  The information in the byte[]
 * needs to be enough to recreate the Token object at the target server.
 * @return byte[]
 */
 public byte[] getBytes ()
 {
  if (hashtable != null)
  {
   try
   {
    // do this if the object is set read-only during login commit,
    // since this guarantees no new data gets set.
    if (isReadOnly() && tokenBytes == null)
     tokenBytes = some_encryption_algorithm (hashtable);

    // you can deserialize the tokenBytes using a similiar decryption algorithm.
    return tokenBytes;
   }
   catch (Exception e)
   {
    e.printStackTrace();
    return null;
   }
  }

  System.out.println("getBytes: returning null");
  return null;
 }

/**
```

```
 * Gets the name of the token, used to identify the byte[] in the protocol message.
 * @return String
 */
 public String getName()
 {
  return "myCookieName";
 }

/**
 * Gets the version of the token as a short.  This is also used to identify the
 * byte[] in the protocol message.
 * @return short
 */
 public short getVersion()
 {
  String[] version = getAttributes("version");

  if (version != null && version[0] != null)
   return new Short(version[0]).shortValue();

  System.out.println("getVersion: returning default of 1");
  return 1;
    }

/**
 * When called, the token becomes irreversibly read-only.  The implementation
 * needs to ensure any setter methods check that this has been set.
 */
 public void setReadOnly()
 {
  addAttribute("readonly", "true");
 }

/**
 * Called internally to see if the token is readonly
 */
 private boolean isReadOnly()
 {
  String[] readonly = getAttributes("readonly");

  if (readonly != null && readonly[0] != null)
   return new Boolean(readonly[0]).booleanValue();

  System.out.println("isReadOnly: returning default of false");
  return false;
 }

/**
 * Gets the attribute value based on the named value.
 * @param String key
 * @return String[]
 */
 public String[] getAttributes(String key)
 {
  ArrayList array = (ArrayList) hashtable.get(key);

  if (array != null && array.size() > 0)
  {
   return (String[]) array.toArray(new String[0]);
  }

  return null;
 }

/**
 * Sets the attribute name/value pair.  Returns the previous values set for key,
 * or null if not previously set.
 * @param String key
 * @param String value
 * @returns String[];
 */
 public String[] addAttribute(String key, String value)
 {
  // get the current value for the key
  ArrayList array = (ArrayList) hashtable.get(key);

  if (!isReadOnly())
  {
   // copy the ArrayList to a String[] as it currently exists
   String[] old_array = null;
   if (array != null && array.size() > 0)
    old_array = (String[]) array.toArray(new String[0]);

   // allocate a new ArrayList if one was not found
   if (array == null)
    array = new ArrayList();

   // add the String to the current array list
   array.add(value);
```

```
   // add the current ArrayList to the Hashtable
   hashtable.put(key, array);

   // return the old array
   return old_array;
  }

  return (String[]) array.toArray(new String[0]);
 }


/**
 * Gets the List of all attribute names present in the token.
 * @return java.util.Enumeration
 */
 public java.util.Enumeration getAttributeNames()
 {
  return hashtable.keys();
 }

/**
 * Returns a deep copying of this token, if necessary.
 * @return Object
 */
 public Object clone()
 {
  com.ibm.websphere.security.token.CustomSingleSignonImpl deep_clone =
   new com.ibm.websphere.security.token.CustomSingleSignonTokenImpl();

  java.util.Enumeration keys = getAttributeNames();

  while (keys.hasMoreElements())
  {
   String key = (String) keys.nextElement();

   String[] list = (String[]) getAttributes(key);

   for (int i=0; i<list.length; i++)
    deep_clone.addAttribute(key, list[i]);
  }

      return deep_clone;
 }
}
```

Example: A custom single sign-on token login module

This file shows how to determine if the login is an initial login or a propagation login.

For information on initialization and on what to do during login and commit, see Developing custom login modules for a system login configuration for JAAS.

```
public customLoginModule()
{
 public void initialize(Subject subject, CallbackHandler callbackHandler,
     Map sharedState, Map options)
 {
  _sharedState = sharedState;
 }

 public boolean login() throws LoginException
 {
     // Handles the WSTokenHolderCallback to see if this is an initial or
     // propagation login.
  Callback callbacks[] = new Callback[1];
  callbacks[0] = new WSTokenHolderCallback("Authz Token List: ");

  try
  {
   callbackHandler.handle(callbacks);
  }
  catch (Exception e)
  {
   // handle exception
  }

  // Receives the ArrayList of TokenHolder objects (the serialized tokens)
  List authzTokenList = ((WSTokenHolderCallback) callbacks[0]).getTokenHolderList();

  if (authzTokenList != null)
  {
   // iterate through the list looking for your custom token
   for (int i=0; i
   for (int i=0; i<authzTokenList.size(); i++)
   {
    TokenHolder tokenHolder = (TokenHolder)authzTokenList.get(i);

    // Looks for the name and version of your custom SingleSignonToken
```

```
        // implementation
   if (tokenHolder.getName().equals("myCookieName")
            && tokenHolder.getVersion() == 1)
   {
    // Passes the bytes into your custom SingleSignonToken constructor
          // to deserialize
    customSSOToken = new
     com.ibm.websphere.security.token.CustomSingleSignonTokenImpl
                (tokenHolder.getBytes());

   }
  }
 }
 else
        // This is not a propagation login. Create a new instance of your
        // SingleSignonToken implementation
 {
     // Gets the principal from the default SingleSignonToken. This principal
     //  must match all tokens.
  defaultAuthToken = (com.ibm.wsspi.security.token.AuthenticationToken)
   sharedState.get(com.ibm.wsspi.security.auth.callback.Constants.WSAUTHTOKEN_KEY);
  String principal = defaultAuthToken.getPrincipal();

  // Adds a new custom single sign-on (SSO) token. This is an initial login.
      //  Pass the principal into the constructor
  customSSOToken = new com.ibm.websphere.security.token.
          CustomSingleSignonTokenImpl(principal);

  // add any initial attributes
  if (customSSOToken != null)
  {
   customSSOToken.addAttribute("key1", "value1");
   customSSOToken.addAttribute("key1", "value2");
   customSSOToken.addAttribute("key2", "value1");
   customSSOToken.addAttribute("key3", "something different");
  }
 }

    // Note: You can add the token to the Subject during commit in case something
    // happens during the login.
}

public boolean commit() throws LoginException
{
 if (customSSOToken != null)
 {
  // Sets the customSSOToken token into the Subject
  try
  {
   public final SingleSignonToken customSSOTokenPriv = customSSOToken;
         // Do this in a doPrivileged code block so that application code does not
         // need to add additional permissions
   java.security.AccessController.doPrivileged(new java.security.PrivilegedAction()
   {
    public Object run()
    {
     try
     {
      // Adds the custom SSO token if it is not null and
               //  not already in the Subject
                              if ((customSSOTokenPriv != null) &&
        (!subject.getPrivateCredentials().
                        contains(customSSOTokenPriv)))
      {
       subject.getPrivateCredentials().
                    add(customSSOTokenPriv);
      }
     }
     catch (Exception e)
     {
      throw new WSLoginFailedException (e.getMessage(), e);
     }

     return null;
    }
   });
  }
  catch (Exception e)
  {
   throw new WSLoginFailedException (e.getMessage(), e);
  }
 }
}

// Defines your login module variables
com.ibm.wsspi.security.token.SingleSignonToken customSSOToken = null;
com.ibm.wsspi.security.token.AuthenticationToken defaultAuthToken = null;
java.util.Map _sharedState = null;
}
```

Example: An HTTP cookie retrieval

The following example shows you how to retrieve a cookie from an HTTP request, decode the cookie so that it is back to your original bytes, and create your custom SingleSignonToken object from the bytes. This example shows how to complete these steps from a login module. However, you also can complete these steps using a servlet.

For information on what to do during initialization, login and commit, see Developing custom login modules for a system login configuration for JAAS.

```
public customLoginModule()
{
 public void initialize(Subject subject, CallbackHandler callbackHandler,
     Map sharedState, Map options)
 {
  _sharedState = sharedState;
 }

 public boolean login() throws LoginException
 {
     // Handles the WSTokenHolderCallback to see if this is an
     // initial or propagation login.
  Callback callbacks[] = new Callback[2];
  callbacks[0] = new WSTokenHolderCallback("Authz Token List: ");
  callbacks[1] = new WSServletRequestCallback("HttpServletRequest: ");

  try
  {
   callbackHandler.handle(callbacks);
  }
  catch (Exception e)
  {
   // Handles the exception
  }

  // receive the ArrayList of TokenHolder objects (the serialized tokens)
  List authzTokenList = ((WSTokenHolderCallback) callbacks[0]).getTokenHolderList();
  javax.servlet.http.HttpServletRequest request =
         ((WSServletRequestCallback) callbacks[1]).getHttpServletRequest();

  if (request != null)
  {

   // Checks if the cookie is present
   javax.servlet.http.Cookie[] cookies = request.getCookies();
   String[] cookieStrings = getCookieValues (cookies, "myCookeName1");

   if (cookieStrings != null)
   {
    String cookieVal = null;
    for (int n=0;n<cookieStrings.length;n++)
    {
     cookieVal = cookieStrings[n];
     if (cookieVal.length()>0)
     {
               // Removes the cookie encoding from the cookie to get
               // your custom bytes
      byte[] cookieBytes =
       com.ibm.websphere.security.WSSecurityHelper.
                     convertCookieStringToBytes(cookieVal);
      customSSOToken =
       new com.ibm.websphere.security.token.
                     CustomSingleSignonTokenImpl(cookieBytes);

               // Now that you have your cookie from the request,
               // you can do something with it here, or add it
               // to the Subject in the commit() method for use later.
      if (debug || tc.isDebugEnabled())
      {
       System.out.println("*** GOT MY CUSTOM SSO TOKEN FROM
                     THE REQUEST ***");
      }
     }
    }
   }
  }

 }

 public boolean commit() throws LoginException
 {
  if (customSSOToken != null)
  {
   // Sets the customSSOToken token into the Subject
   try
   {
```

```
    public final SingleSignonToken customSSOTokenPriv = customSSOToken;
          // Do this in a doPrivileged code block so that application code does not
          // need to add additional permissions
    java.security.AccessController.doPrivileged(new java.security.PrivilegedAction()
    {
     public Object run()
     {
      try
      {
                 // Add the custom SSO token if it is not null and not
                 // already in the Subject
                               if ((customSSOTokenPriv != null) &&
         (!subject.getPrivateCredentials().
                         contains(customSSOTokenPriv)))
       {
        subject.getPrivateCredentials().add(customSSOTokenPriv);
       }
      }
      catch (Exception e)
      {
       throw new WSLoginFailedException (e.getMessage(), e);
      }

      return null;
     }
    });
   }
   catch (Exception e)
   {
    throw new WSLoginFailedException (e.getMessage(), e);
   }
  }
 }

 // Private method to get the specific cookie from the request
 private String[] getCookieValues (Cookie[] cookies, String hdrName)
 {
  Vector retValues = new Vector();
  int numMatches=0;
  if (cookies != null)
  {
   for (int i = 0; i < cookies.length; ++i)
   {
    if (hdrName.equals(cookies[i].getName()))
    {
     retValues.add(cookies[i].getValue());
     numMatches++;
     System.out.println(cookies[i].getValue());
    }
   }
  }

  if (retValues.size()>0)
   return (String[]) retValues.toArray(new String[numMatches]);
  else
   return null;
 }

 // Defines your login module variables
 com.ibm.wsspi.security.token.SingleSignonToken customSSOToken = null;
 com.ibm.wsspi.security.token.AuthenticationToken defaultAuthToken = null;
 java.util.Map _sharedState = null;
}
```

Implementing a custom authentication token for security attribute propagation

This topic explains how you might create your own authentication token implementation, which is set in the login Subject and propagated downstream.

About this task

With this implementation you can specify an authentication token that can be used by a custom login module or application. Consider writing your own implementation if you want to accomplish one of the following tasks:

- Isolate your attributes within your own implementation.
- Serialize the information using custom serialization. You must deserialize the bytes at the target and add that information back on the thread. This task also might include encryption and decryption.

- Affect the overall uniqueness of the Subject using the getUniqueID application programming interface (API).

Important: Custom authentication token implementations are not used by the security runtime in WebSphere Application Server to enforce authentication. WebSphere Application Security runtime uses this token in the following situations only:

- Call the getBytes method for serialization
- Call the getForwardable method to determine whether to serialize the authentication token.
- Call the getUniqueId method for uniqueness
- Call the getName and the getVersion methods for adding serialized bytes to the token holder that is sent downstream

All of the other uses are custom implementations.

To implement a custom authentication token, you must complete the following steps:

Procedure

1. Write a custom implementation of the AuthenticationToken interface. Many different methods are available for implementing the AuthenticationToken interface. However, make sure the methods that are required by the AuthenticationToken interface and the token interface are fully implemented. After you implement this interface, you can place it in the *app_server_root*/classes directory. Alternatively, you can place the class in any private directory. However, make sure that the WebSphere Application Server class loader can locate the class and that it is granted the appropriate permissions. You can add the Java archive (JAR) file or directory that contains this class into the server.policy file so the class has the necessary permissions required by the server code.

   Tip: All of the token types that are defined by the propagation framework have similar interfaces. The token types are marker interfaces that implement the com.ibm.wsspi.security.token.Token interface. This interface defines most of the methods. If you plan to implement more than one token type, consider creating an abstract class that implements the com.ibm.wsspi.security.token.Token interface. All of your token implementations, including the authentication token, might extend the abstract class and then most of the work is complete.

   To see an implementation of the AuthenticationToken interface, see “Example: A com.ibm.wsspi.security.token.AuthenticationToken implementation” on page 911.
2. Add and receive the custom authentication token during WebSphere Application Server logins. This task is typically accomplished by adding a custom login module to the various application and system login configurations. However, to deserialize the information you must plug in a custom login module. After the object is instantiated in the login module, you can add the object to the Subject during the commit method.

   If you only want to add information to the Subject to get propagated, see “Propagating a custom Java serializable object for security attribute propagation” on page 916. If you want to ensure that the information is propagated, do your own custom serialization, or specify the uniqueness for Subject caching purposes, consider writing your own authentication token implementation.

   The code sample in “Example: A custom authentication token login module” on page 915, shows how to determine if the login is an initial login or a propagation login. The difference between these login types is whether the WSTokenHolderCallback callback contains propagation data. If the callback does not contain propagation data, initialize a new custom authentication token implementation and set it into the Subject. If the callback contains propagation data, look for your specific custom authentication token TokenHolder instance, convert the byte array back into your custom AuthenticationToken object, and set it back into the Subject. The code sample shows both instances.

   You can make your authentication token read-only in the commit phase of the login module. If you do not make the token read-only, attributes can be added within your applications.

3. Add your custom login module to WebSphere Application Server system login configurations that already contain the com.ibm.ws.security.server.lm.wsMapDefaultInboundLoginModule login module for receiving serialized versions of your custom authorization token.

   Because this login module relies on information in the shared state that is added by the com.ibm.ws.security.server.lm.wsMapDefaultInboundLoginModule login module, add this login module after the com.ibm.ws.security.server.lm.wsMapDefaultInboundLoginModule login module. For information on how to add your custom login module to the existing login configurations, see Developing custom login modules for a system login configuration for JAAS.

Results

After completing these steps, you have implemented a custom authentication token.

Example: A com.ibm.wsspi.security.token.AuthenticationToken implementation

The following example illustrates an authentication token implementation. The following sample code does not extend an abstract class, but rather implements the com.ibm.wsspi.security.token.AuthenticationToken interface directly. You can implement the interface directly, but it might cause you to write duplicate code. However, you might choose to implement the interface directly if considerable differences exist between how you handle the various token implementations.

```
package com.ibm.websphere.security.token;

import com.ibm.websphere.security.WSSecurityException;
import com.ibm.websphere.security.auth.WSLoginFailedException;
import com.ibm.wsspi.security.token.*;
import com.ibm.websphere.security.WebSphereRuntimePermission;
import java.io.ByteArrayOutputStream;
import java.io.ByteArrayInputStream;
import java.io.DataOutputStream;
import java.io.DataInputStream;
import java.io.ObjectOutputStream;
import java.io.ObjectInputStream;
import java.io.OutputStream;
import java.io.InputStream;
import java.util.ArrayList;

public class CustomAuthenticationTokenImpl implements com.ibm.wsspi.security.
   token.AuthenticationToken
{
 private java.util.Hashtable hashtable = new java.util.Hashtable();
 private byte[] tokenBytes = null;
  // 2 hours in millis, by default
   private static long expire_period_in_millis = 2*60*60*1000;
 private String oidName = "your_oid_name";
  // This string can really be anything if you do not want to use an OID.

/**
 * Constructor used to create initial AuthenticationToken instance
 */
 public CustomAuthenticationTokenImpl (String principal)
 {
  // Sets the principal in the token
  addAttribute("principal", principal);
  // Sets the token version
  addAttribute("version", "1");
  // Sets the token expiration
  addAttribute("expiration", new Long(System.currentTimeMillis()
        + expire_period_in_millis).toString());
 }

/**
 * Constructor used to deserialize the token bytes received during a
 * propagation login.
 */
 public CustomAuthenticationTokenImpl (byte[] token_bytes)
 {
  try
  {
       // The data in token_bytes should be signed and encrypted if the
       // hashtable is acting as an authentication token.
   hashtable = (java.util.Hashtable) custom_decryption_algorithm (token_bytes);
  }
  catch (Exception e)
  {
   e.printStackTrace();
  }
 }

/**
```

```
 * Validates the token including expiration, signature, and so on.
 * @return boolean
 */

 public boolean isValid ()
 {
  long expiration = getExpiration();

  // If you set the expiration to 0, the token does not expire
  if (expiration != 0)
  {
   // Returns a response that identifies whether this token is still valid
   long current_time = System.currentTimeMillis();

   boolean valid = ((current_time < expiration) ? true : false);
   System.out.println("isValid: returning " + valid);
   return valid;
  }
  else
  {
   System.out.println("isValid: returning true by default");
   return true;
  }
 }

/**
 * Gets the expiration as a long type.
 * @return long
 */
 public long getExpiration()
 {
  // Gets the expiration value from the hashtable
  String[] expiration = getAttributes("expiration");

  if (expiration != null && expiration[0] != null)
  {
   // The expiration is the first element and there should only be one expiration
   System.out.println("getExpiration: returning " + expiration[0]);
   return new Long(expiration[0]).longValue();
  }

  System.out.println("getExpiration: returning 0");
  return 0;
 }

/**
 * Returns if this token should be forwarded/propagated downstream.
 * @return boolean
 */
 public boolean isForwardable()
 {
     // You can choose whether your token gets propagated. In some cases
     // you might want it to be local only.
  return true;
 }

/**
 * Gets the principal to which this token belongs. If this is an
 * authorization token, this principal string must match the
 * authentication token principal string or the message is rejected.
 * @return String
 */
 public String getPrincipal()
 {
  // This value might be any combination of attributes
  String[] principal = getAttributes("principal");

  if (principal != null && principal[0] != null)
  {
   return principal[0];
  }

  System.out.println("getExpiration: returning null");
  return null;
 }

/**
 * Returns a unique identifier of the token based upon information the provider
 * considers makes this a unique token. This identifier is used for caching purposes
 * and can be used in combination with other token unique IDs that are part of
 * the same Subject.
 *
 * This method should return null if you want the accessID of the user to represent
 * uniqueness.  This is the typical scenario.
 *
 * @return String
 */
 public String getUniqueID()
 {
     // If you do not want to affect the cache lookup, just return NULL here.
```

```
    return null;

  String cacheKeyForThisToken = "dynamic attributes";

    // If you do want to affect the cache lookup, return a string of
    // attributes that you want factored into the lookup.
  return cacheKeyForThisToken;
 }

/**
 * Gets the bytes to be sent across the wire. The information in the byte[]
 * needs to be enough to recreate the token object at the target server.
 * @return byte[]
 */
 public byte[] getBytes ()
 {
  if (hashtable != null)
  {
   try
   {
          // Do this if the object is set read-only during login commit
          // because this ensures that new data is not set.
    if (isReadOnly() && tokenBytes == null)
     tokenBytes = custom_encryption_algorithm (hashtable);

    return tokenBytes;
   }
   catch (Exception e)
   {
    e.printStackTrace();
    return null;
   }
  }

  System.out.println("getBytes: returning null");
  return null;
 }

/**
 * Gets the name of the token, which is used to identify the byte[] in the
 * protocol message.
 * @return String
 */
 public String getName()
 {
  return oidName;
 }

/**
 * Gets the version of the token as a short type.  This also is used
 * to identify the byte[] in the protocol message.
 * @return short
 */
 public short getVersion()
 {
  String[] version = getAttributes("version");

  if (version != null && version[0] != null)
   return new Short(version[0]).shortValue();

  System.out.println("getVersion: returning default of 1");
  return 1;
    }

/**
 * When called, the token becomes irreversibly read-only.  The implementation
 * needs to ensure that any set methods check that this state has been set.
 */
 public void setReadOnly()
 {
  addAttribute("readonly", "true");
 }

/**
 * Called internally to see if the token is read-only
 */
 private boolean isReadOnly()
 {
  String[] readonly = getAttributes("readonly");

  if (readonly != null && readonly[0] != null)
   return new Boolean(readonly[0]).booleanValue();

  System.out.println("isReadOnly: returning default of false");
  return false;
 }

/**
 * Gets the attribute value based on the named value.
 * @param String key
```

```
 * @return String[]
 */
 public String[] getAttributes(String key)
 {
  ArrayList array = (ArrayList) hashtable.get(key);

  if (array != null && array.size() > 0)
  {
   return (String[]) array.toArray(new String[0]);
  }

  return null;
 }

/**
 * Sets the attribute name/value pair.  Returns the previous values set for key,
 * or null if not previously set.
 * @param String key
 * @param String value
 * @returns String[];
 */
 public String[] addAttribute(String key, String value)
 {
  // Gets the current value for the key
  ArrayList array = (ArrayList) hashtable.get(key);

  if (!isReadOnly())
  {
   // Copies the ArrayList to a String[] as it currently exists
   String[] old_array = null;
   if (array != null && array.size() > 0)
    old_array = (String[]) array.toArray(new String[0]);

   // Allocates a new ArrayList if one was not found
   if (array == null)
    array = new ArrayList();

   // Adds the String to the current array list
   array.add(value);

   // Adds the current ArrayList to the Hashtable
   hashtable.put(key, array);

   // Returns the old array
   return old_array;
  }

  return (String[]) array.toArray(new String[0]);
 }


/**
 * Gets the list of all attribute names present in the token.
 * @return java.util.Enumeration
 */
 public java.util.Enumeration getAttributeNames()
 {
  return hashtable.keys();
 }

/**
 * Returns a deep copying of this token, if necessary.
 * @return Object
 */
 public Object clone()
 {
  com.ibm.wsspi.security.token.AuthenticationToken deep_clone =
   new com.ibm.websphere.security.token.CustomAuthenticationTokenImpl();

  java.util.Enumeration keys = getAttributeNames();

  while (keys.hasMoreElements())
  {
   String key = (String) keys.nextElement();

   String[] list = (String[]) getAttributes(key);

   for (int i=0; i<list.length; i++)
    deep_clone.addAttribute(key, list[i]);
  }

      return deep_clone;
 }

/**
 * This method returns true if this token is storing a user ID and password
 * instead of a token.
 * @return boolean
 */
 public boolean isBasicAuth()
```

```
 {
  return false;
 }
}
```

Example: A custom authentication token login module

This examples shows how to determine if the login is an initial login or a propagation login.

For information on what to do during initialization, login and commit, see “Developing custom login modules for a system login configuration for JAAS” on page 442.

```
public customLoginModule()
{
 public void initialize(Subject subject, CallbackHandler callbackHandler,
    Map sharedState, Map options)
 {
  _sharedState = sharedState;
 }

 public boolean login() throws LoginException
 {
  // Handles the WSTokenHolderCallback to see if this is an initial or
     // propagation login.
  Callback callbacks[] = new Callback[1];
  callbacks[0] = new WSTokenHolderCallback("Authz Token List: ");

  try
  {
   callbackHandler.handle(callbacks);
  }
  catch (Exception e)
  {
   // Handles exception
  }

  // Receives the ArrayList of TokenHolder objects (the serialized tokens)
  List authzTokenList = ((WSTokenHolderCallback) callbacks[0]).getTokenHolderList();

  if (authzTokenList != null)
  {
   // Iterates through the list looking for your custom token
   for (int i=0; i<authzTokenList.size(); i++)
   {
    TokenHolder tokenHolder = (TokenHolder)authzTokenList.get(i);

          // Looks for the name and version of your custom AuthenticationToken
          // implementation
    if (tokenHolder.getName().equals("your_oid_name") && tokenHolder.getVersion() == 1)
    {
           // Passes the bytes into your custom AuthenticationToken constructor
           // to deserialize
     customAuthzToken = new
      com.ibm.websphere.security.token.
                CustomAuthenticationTokenImpl(tokenHolder.getBytes());

    }
   }
  }
  else
          // This is not a propagation login. Create a new instance of your
          // AuthenticationToken implementation
  {
        //  Gets the principal from the default AuthenticationToken. This principal
        //  should match all default tokens.
        //  Note: WebSphere Application Server runtime only enforces this for
        //  default tokens. Thus, you can choose
        //  to do this for custom tokens, but it is not required.
   defaultAuthToken = (com.ibm.wsspi.security.token.AuthenticationToken)
    sharedState.get(com.ibm.wsspi.security.auth.callback.Constants.WSAUTHTOKEN_KEY);
   String principal = defaultAuthToken.getPrincipal();

       // Adds a new custom authentication token. This is an initial login. Pass
       // the principal into the constructor
   customAuthToken = new com.ibm.websphere.security.token.
          CustomAuthenticationTokenImpl(principal);

   // Adds any initial attributes
   if (customAuthToken != null)
   {
    customAuthToken.addAttribute("key1", "value1");
    customAuthToken.addAttribute("key1", "value2");
    customAuthToken.addAttribute("key2", "value1");
    customAuthToken.addAttribute("key3", "something different");
   }
  }

     // Note: You can add the token to the Subject during commit in case
```

```
    // something happens during the login.
}

public boolean commit() throws LoginException
{
 if (customAuthToken != null)
 {
  // Sets the customAuthToken token into the Subject
  try
  {
   private final AuthenticationToken customAuthTokenPriv = customAuthToken;
         // Do this in a doPrivileged code block so that application code does
         // not need to add additional permissions
   java.security.AccessController.doPrivileged(new java.security.PrivilegedAction()
   {
    public Object run()
    {
     try
     {
                 // Adds the custom Authentication token if it is not
                 // null and not already in the Subject
                              if ((customAuthTokenPriv != null) &&
        (!subject.getPrivateCredentials().
                       contains(customAuthTokenPriv)))
      {
       subject.getPrivateCredentials().add(customAuthTokenPriv);
      }
     }
     catch (Exception e)
     {
      throw new WSLoginFailedException (e.getMessage(), e);
     }

     return null;
    }
   });
  }
  catch (Exception e)
  {
   throw new WSLoginFailedException (e.getMessage(), e);
  }
 }
}

// Defines your login module variables
com.ibm.wsspi.security.token.AuthenticationToken customAuthToken = null;
com.ibm.wsspi.security.token.AuthenticationToken defaultAuthToken = null;
java.util.Map _sharedState = null;
}
```

Propagating a custom Java serializable object for security attribute propagation

This document describes how to add an object into the Subject from a login module and describes other infrastructure considerations to make sure that the Java object gets propagated.

Before you begin

Prior to completing this task, verify that security propagation is enabled in the administrative console.

About this task

With security attribute propagation enabled, you can propagate data either horizontally with single sign-on (SSO) enabled or downstream using Common Secure Interoperability Version 2 (CSIv2). When a login occurs, either through an application login configuration or a system login configuration, a custom login module can be plugged in to add Java serialized objects into the Subject during login. This document describes how to add an object into the Subject from a login module and describes other infrastructure considerations to make sure that the Java object gets propagated.

Procedure

1. Add your custom Java object into the Subject from a custom login module. A two-phase process exists for each Java Authentication and Authorization Service (JAAS) login module. WebSphere Application Server completes the following processes for each login module present in the configuration:

login method

In this step, the login configuration callbacks are analyzed, if necessary, and the new objects or credentials are created.

commit method

In this step, the objects or credentials that are created during login are added into the Subject.

After a custom Java object is added into the Subject, WebSphere Application Server serializes the object on the sending server, deserializes the object on the receiving server, and adds the object back into the Subject downstream. However, some requirements exist for this process to occur successfully. For more information on the JAAS programming model, see the JAAS information provided in the Security: Resources for learning article.

Important: Whenever you plug a custom login module into the login infrastructure of WebSphere Application Server, make sure that the code is trusted. When you put the classes together in a Java archive (JAR) file and add the file to the *app_server_root*/lib/ext/ directory, the login module has Java 2 Security AllPermissions permissions . It is recommended that you add your login module and other infrastructure classes into any private directory. However, you must modify the *profile_root*/properties/server.policy file to make sure that your private directory, Java archive (JAR) file, or both have the permissions required to run the application programming interfaces (API) that are called from the login module. Because the login module might be run after the application code on the call stack, you might add doPrivileged code so that you do not need to add additional properties to your applications.

The following code sample shows how to add doPrivileged code. For information on what to do during initialization, login and commit, see “Developing custom login modules for a system login configuration for JAAS” on page 442.

```
public customLoginModule()
{
 public void initialize(Subject subject, CallbackHandler callbackHandler,
     Map sharedState, Map options)
 {

 }

 public boolean login() throws LoginException
 {
     // Construct callback for the WSTokenHolderCallback so that you
     // can determine if
     // your custom object has propagated
    Callback callbacks[] = new Callback[1];
    callbacks[0] = new WSTokenHolderCallback("Authz Token List: ");

    try
    {
         _callbackHandler.handle(callbacks);
    }
    catch (Exception e)
    {
    throw new LoginException (e.getLocalizedMessage());
    }

    // Checks to see if any information is propagated into this login
    List authzTokenList = ((WSTokenHolderCallback) callbacks[1]).
            getTokenHolderList();

    if (authzTokenList != null)
    {
        for (int i = 0; i< authzTokenList.size(); i++)
             {
            TokenHolder tokenHolder = (TokenHolder)authzTokenList.get(i);

                  //  Look for your custom object. Make sure you use
                  //  "startsWith"because there is some data appended
                  //  to the end of the name indicating in which Subject
                  //  Set it belongs. Example from getName():
                  //  "com.acme.CustomObject (1)". The class name is
                  //  generated at the sending side by calling the
                  //  object.getClass().getName() method. If this object
                  //  is deserialized by WebSphere Application Server,
                  //  then return it and you do not need to add it here.
                  //  Otherwise, you can add it below.
                  //  Note: If your class appears in this list and does
                  //  not use custom serialization (for example, an
                  //  implementation of the Token interface described in
```

```
                //  the Propagation Token Framework), then WebSphere
                //  Application  Server automatically deserializes the
                //  Java object for you. You might just return here if
                //  it is found in the list.

           if (tokenHolder.getName().startsWith("com.acme.CustomObject"))
         return true;
      }
   }
      //  If you get to this point, then your custom object has not propagated
       myCustomObject = new com.acme.CustomObject();
       myCustomObject.put("mykey", "mydata");
}

public boolean commit() throws LoginException
{
 try
 {
      // Assigns a reference to a final variable so it can be used in
      // the doPrivileged block
  final com.acme.CustomObject myCustomObjectFinal = myCustomObject;
  // Prevents your applications from needing a JAAS getPrivateCredential
      // permission.
  java.security.AccessController.doPrivileged(new java.security.
          PrivilegedExceptionAction()
  {
   public Object run() throws java.lang.Exception
   {
           // Try not to add a null object to the Subject or an object
           // that already exists.
    if (myCustomObjectFinal != null && !subject.getPrivateCredentials().
               contains(myCustomObjectFinal))
    {
              // This call requires a special Java 2 Security permission,
              // see the JAAS application programming interface (API)
              // documentation.
     subject.getPrivateCredentials().add(myCustomObjectFinal);
    }
    return null;
   }
  });
 }
 catch (java.security.PrivilegedActionException e)
 {
  // Wraps the exception in a WSLoginFailedException
  java.lang.Throwable myException = e.getException();
  throw new WSLoginFailedException (myException.getMessage(), myException);
 }
}

// Defines your login module variables
com.acme.CustomObject myCustomObject = null;
}
```

2. Verify that your custom Java class implements the java.io.Serializable interface. An object that is added to the Subject must be serialized if you want the object to propagate. For example, the object must implement the java.io.Serializable interface. If the object is not serialized, the request does not fail, but the object does not propagate. To make sure an object that is added to the Subject is propagated, implement one of the token interfaces that is defined in topics about security attribute propagation or add attributes to one of the following existing default token implementations:

   AuthorizationToken
   Add attributes if they are user-specific.

   PropagationToken
   Add attributes that are specific to an invocation.

   If you are careful adding custom objects and follow all the steps to make sure that WebSphere Application Server can serialize and deserialize the object at each hop, then it is sufficient to use custom Java objects only.

3. Verify that your custom Java class exists on all of the systems that might receive the request.

   When you add a custom object into the Subject and expect WebSphere Application Server to propagate the object, put the class definitions together in a Java archive (JAR) file and add the file to the *app_server_root*/lib/ext/ directory on all of the nodes where serialization or deserialization might occur. Also, verify that the Java class versions are the same.

4. Verify that your custom login module is configured in all of the login configurations used in your environment where you need to add your custom object during a login. Any login configuration that interacts with WebSphere Application Server generates a Subject that might be propagated outbound

for an Enterprise JavaBeans (EJB) request. If you want WebSphere Application Server to propagate a custom object in all cases, make sure that the custom login module is added to every login configuration that is used in your environment. For more information, see “Developing custom login modules for a system login configuration for JAAS” on page 442.

5. Verify that security attribute propagation is enabled on all of the downstream servers that receive the propagated information. When an EJB request is sent to a downstream server and security attribute propagation is disabled on that server, only the authentication token is sent for backwards compatibility. Therefore, you must review the configuration to verify that propagation is enabled in all of the cells that might receive requests. You must check several places in the administrative console to make sure propagation is fully enabled.
6. Add any custom objects to the propagation exclude list that you do not want to propagate. You can configure a property to exclude the propagation of objects that match specific class names, package names, or both. For example, you can have a custom object that is related to a specific process. If the object is propagated, it does not contain valid information. You must tell WebSphere Application Server not to propagate this object. Complete the following steps to specify the object in the propagation exclude list, using the administrative console:
   a. Click **Security > Global security > Custom properties > New**.
   b. Add `com.ibm.ws.security.propagationExcludeList` in the **Name** field.
   c. Add the name of the custom object in the **Value** field. You can add a list of custom objects to the propagation exclude list, separated by a colon (:). For example, you might enter `com.acme.CustomLocalObject:com.acme.private.*`. You can enter a class name such as `com.acme.CustomLocalObject` or a package name such as `com.acme.private.*`. In this example, WebSphere Application Server does not propagate any class that equals `com.acme.CustomLocalObject` or begins with `com.acme.private.`

      Although you can add custom objects to the propagation exclude list, you must be aware of a side effect. WebSphere Application Server stores the opaque token, or the serialized Subject contents, in a local cache for the life of the single sign-on (SSO) token. The life of the SSO token, which has a default of two hours, is configured in the SSO properties on the administrative console. The information that is added to the opaque token includes only the objects not in the exclude list.

      Ensure that your SSO token timeout value is greater that the authentication cache timeout value. To modify the authentication cache, see the documentation about the authentication cache settings.

Results

As a result of this task, custom Java serializable objects are propagated horizontally or downstream. For more information on the differences between horizontal and downstream propagation, see topics about security attribute propagation or add attributes to one of the following existing default token implementations:.

Developing a custom interceptor for trust associations

You can define the interceptor class method that you want to use. WebSphere Application Server supports two trust association interceptor interfaces: com.ibm.wsspi.security.TrustAssociationInterceptor and com.ibm.wsspi.security.tai.TrustAssociationInterceptor.

Before you begin

If you are using a third party reverse proxy server other than Tivoli WebSEAL, you must provide an implementation class for the product interceptor interface for your proxy server. This article describes the com.ibm.wsspi.security.TrustAssociationInterceptor.java interface that you must implement.

Note: The Trust Association Interceptor (TAI) interface (com.ibm.wsspi.security.tai.TrustAssociationInterceptor) supports several new features and is different from the existing com.ibm.wsspi.security.TrustAssociationInterceptor interface.

Procedure

1. Define the interceptor class method. WebSphere Application Server provides the interceptor Java interface, com.ibm.wsspi.security.TrustAssociationInterceptor, which defines the following methods:
   - **`public boolean isTargetInterceptor(HttpServletRequest req)`** creates `WebTrustAssociationException;`.

     The isTargetInterceptor method determines whether the request originated with the proxy server associated with the interceptor. The implementation code must examine the incoming request object and determine if the proxy server forwarding the request is a valid proxy server for this interceptor. The result of this method determines whether the interceptor processes the request or not.
   - **`public void validateEstablishedTrust (HttpServletRequest req)`** creates `WebTrustAssociationException;`.

     The validateEstablishedTrust method determines if the proxy server from which the request originated is trusted or not. This method is called after the isTargetInterceptor method. The implementation code must authenticate the proxy server. The authentication mechanism is proxy-server specific. For example, in the product implementation for the WebSEAL server, this method retrieves the basic authentication information from the HTTP header and validates the information against the user registry used by WebSphere Application Server. If the credentials are invalid, the code creates the WebTrustAssociationException, indicating that the proxy server is not trusted and the request is to be denied.
   - **`public String getAuthenticatedUsername(HttpServletRequest req)`** creates `WebTrustAssociationException;`.

     The getAuthenticatedUsername method is called after trust is established between the proxy server and WebSphere Application Server. The product has accepted the proxy server authentication of the request and must now authorize the request. To authorize the request, the name of the original requestor must be subjected to an authorization policy to determine if the requestor has the necessary privilege. The implementation code for this method must extract the user name from the HTTP request header and determine if that user is entitled to the requested resource. For example, in the product implementation for the WebSEAL server, the method looks for an `iv-user` attribute in the HTTP request header and extracts the user ID associated with it for authorization.
2. Configuring the interceptor. To make an interceptor configurable, the interceptor must extend com.ibm.wsspi.security.WebSphereBaseTrustAssociationInterceptor. Implement the following methods:
   `public int init (java.util.Properties props);`

   The init(Properties) method accepts a `java.util.Properties` object, which contains the set of properties required to initialize the interceptor. All the properties set for an interceptor (by using the **Custom Properties** link for that interceptor or using scripting) is sent to this method. The interceptor then can use these properties to initialize itself. For example, in the product implementation for the WebSEAL server, this method reads the hosts and ports so that a request coming in can be verified to originate from trusted hosts and ports. A return value of **0** implies that the interceptor initialization is successful. Any other value implies that the initialization is not successful and the interceptor is ignored.

   Applicability of the following list

   If a previous implementation of the trust association interceptor returns a different error status you can either change your implementation to match the expectations or make one of the following changes:
   - Add the com.ibm.wsspi.security.trustassociation.initStatus property in the trust association interceptor custom properties. Set the property to the value that indicates that the interceptor is successfully initialized. All of the other possible values imply failure. In case of failure, the corresponding trust association interceptor is not used.
   - Add the com.ibm.wsspi.security.trustassociation.ignoreInitStatus property in the trust association interceptor custom properties. Set the value of this property to **true**, which tells WebSphere Application Server to ignore the status of this method. If you add this property to the custom properties, WebSphere Application Server does not check the return status, which is similar to previous versions of WebSphere Application Server.

public void cleanup ();
This method is called when the application server is stopped. It is used to prepare the interceptor for termination.

public void setVersion (String s);
This method is optional. The method is used to set the version and is for informational purpose only. The default value is `Unspecified`.

You must configure the following methods implemented by the custom interceptor implementation. **This listing only shows the methods and does not include any implementation.**

```
*********************************************************************
import java.util.*;
import javax.servlet.http.HttpServletRequest;
import com.ibm.websphere.security.*;

public class myTAIImpl extends WebSphereBaseTrustAssociationInterceptor
      implements TrustAssociationInterceptor
{

      public myTAIImpl ()
      {
      }


      public boolean isTargetInterceptor (HttpServletRequest req)
          throws WebTrustAssociationException
      {

           //return true if this is the target interceptor, else return false.
      }


      public TAIResult negotiateValidateandEstablishTrust (HttpServletRequest req, HttpServletResponse res)
           throws WebTrustAssociationFailedException
      {
           //validate the request and establish trust.
           //create and return the TAIResult
public int initialize (Properties props)
      {
           //initialize the implementation. If successful return 0, else return 1.
      }

     public String getVersion()
     {
        //Return version
     }

   public String getType()
     {
        //Return type
     }

      public void cleanup ()
      {
           //Cleanup code.

      }


}
```

Note: If the init(Properties) method is implemented as described previously in your custom interceptor, this note does not apply to your implementation, and you can move on to the next step. Previous versions of com.ibm.wsspi.security.WebSphereBaseTrustAssociationInterceptor include the public int init (String propsfile) method. This method is no longer required since the interceptor properties are not read from a file. The properties are now entered in the administrative console **Custom Properties** link of the interceptor using the administrative console or scripts. These properties then are made available to your implementation in the

init(Properties) method. However, for backward compatibility, the init(String) method still is supported. The init(String) method is called by the default implementation of init(Properties) as shown in the following example.

```
// Default implementation of init(Properties props) method. A Custom
   // implementation should override this.
   public int init (java.util.Properties props)
   {
      String type =
       props.getProperty("com.ibm.wsspi.security.trustassociation.types");
      String classfile=
       props.getProperty("com.ibm.wsspi.security.trustassociation."
       +type+".config");
      if (classfile != null && classfile.length() > 0 ) {
         return init(classfile);
      } else {
         return -1;
      }
   }
```

Change your implementation to implement the init(Properties) method instead of relying on init(String propsfile) method. As shown in the previous example, this default implementation reads the properties to load the property file. The com.ibm.wsspi.security.trustassociation.types property gets the file containing the properties by concatenating `.config` to its value.

Note: The init(String) method still works if you want to use it instead of implementing the init(Properties) method. The only requirement is that the file name containing the custom trust association properties should now be entered using the **Custom Properties** link of the interceptor in the administrative console or by using scripts. You can enter the property using *either* of the following methods. The first method is used for backward compatibility with previous versions of WebSphere Application Server.

Method 1:

The same property names used in the previous release are used to obtain the file name. The file name is obtained by concatenating the `.config` to the com.ibm.wsspi.security.trustassociation.types property value.

If the file name is called `myTAI.properties` and is located in the *app_server_root*`/properties` directory, set the following properties:

- com.ibm.wsspi.security.trustassociation.types = myTAItype
- com.ibm.wsspi.security.trustassociation.myTAItype.config = *app_server_root*/properties/myTAI.properties

Method 2:

You can set the com.ibm.wsspi.security.trustassociation.initPropsFile property in the trust association custom properties to the location of the file. For example, set the following property:

```
com.ibm.wsspi.security.trustassociation.initPropsFile=
app_server_root/properties/myTAI.properties
```

Type the previous code as one continuous line.

The location of the properties file is fully qualified (for example, *app_server_root*`/properties/myTAI.properties`). Because the location can be different in a WebSphere Application Server, Network Deployment environment, use variables such as `${USER_INSTALL_ROOT}` to refer to the WebSphere Application Server installation directory. For example, if the file name is called `myTAI.properties` and it is located in the *app_server_root*`/properties` directory, then set the following properties:

3. Compile the implementation once you have implemented it. For example, *app_server_root*`/java/bin/javac -classpath` *install_root*`/plugins/com.ibm.ws.runtime.jar;<install_root>/dev/JavaEE/j2ee.jar myTAIImpl.java`
   a. Identify the trust association interceptor class file for use when the server is restarted. Place the file either at the*app_server_root/classes* directory OR use the Java Virtual Machine (JVM) system property, `-Dws.ext.dirs` to specify where the file resides.
   b. Restart all the servers.

4. Delete the default WebSEAL interceptor in the administrative console and click **New** to add your custom interceptor. Verify that the class name is dot separated and appears in the class path.
5. Click the **Custom Properties** link to add additional properties that are required to initialize the custom interceptor. These properties are passed to the init(Properties) method of your implementation when it extends the com.ibm.wsspi.security.WebSphereBaseTrustAssociationInterceptor as described in the previous step.
6. Save and synchronize (if applicable) the configuration.
7. Restart the servers for the custom interceptor to take effect.

Example

Refer to the Security: Resources for Learning article for a reference to an example of a custom interceptor.

Trust association interceptor support for Subject creation

The trust association interceptor (TAI) com.ibm.wsspi.security.tai.TrustAssociationInterceptor interface supports several features that are different from the existing com.ibm.websphere.security.TrustAssociationInterceptor interface.

The TAI interface supports a multiphase, negotiated authentication process. For example, some systems require a challenge response protocol back to the client. The two key methods in this interface are:

Key method name

public boolean isTargetInterceptor (HttpServletRequest req)

The isTargetInterceptor method determines whether the request originated with the proxy server that is associated with the interceptor. The implementation code must examine the incoming request object and determine if the proxy server that forwards the request is a valid proxy server for this interceptor. The result of this method determines whether the interceptor processes the request.

Method result

A `true` value tells WebSphere Application Server to have the TAI handle the request.

A `false` value, tells WebSphere Application Server to ignore the TAI.

Key method name

public TAIResult negotiateValidateandEstablishTrust (HttpServletRequest req, HttpServletResponse res)

The negotiateValidateandEstablishTrust method determines whether to trust the proxy server from which the request originated. The implementation code must authenticate the proxy server. The authentication mechanism is proxy-server specific. For example, in the product implementation for the WebSEAL server, this method retrieves the basic authentication information from the HTTP header and validates the information against the user registry that WebSphere Application Serve uses. If the credentials are not valid, the code creates the WebTrustAssociationException exception, which indicates that the proxy server is not trusted and the request is denied. If the credentials are valid, the code returns a TAIResult result, which indicates the status of the request processing with the client identity (Subject and principal name) to use for authorizing the web resource.

Method result

Returns a TAIResult result, which indicates the status of the request processing. You can query the Request object and modify the Response object can be modified.

The TAIResult class has three static methods for creating a TAIResult result. The TAIResult create methods take an int type as the first parameter. WebSphere Application Server expects the result to be a valid HTTP request return code and is interpreted in one of the following ways:

- If the value is `HttpServletResponse.SC_OK`, this response tells WebSphere Application Server that the TAI completed its negotiation. The response also tells WebSphere Application Server to use the information in the TAIResult result to create a user identity.
- Other values tell WebSphere Application Server to return the TAI output, which is placed into the HttpServletResponse response, to the web client. Typically, the web client provides additional information and then places another call to the TAI.

Table 125. TAIResults definitions. The created TAIResults results have the following meanings:

TAIResult	Explanation
public static TAIResult create(int status);	Indicates a status to WebSphere Application Server. The status cannot be SC_OK because the identity information is provided.
public static TAIResult create(int status, String principal);	Indicates a status to WebSphere Application Server and provides the user ID or the unique ID for this user. WebSphere Application Server creates credentials by querying the user registry.
public static TAIResult create(int status, String principal, Subject subject);	Indicates a status to WebSphere Application Server, the user ID or the unique ID for the user, and a custom Subject. If the Subject contains a hashtable, the principal is ignored. The contents of the Subject become part of the eventual user Subject.

All of the following examples are within the negotiateValidateandEstablishTrust method of a TAI.

The following code sample indicates that additional negotiation is required:

```
// Modify the HttpServletResponse object
//  The response code is meaningful only on the client
 return TAIResult.create(HttpServletResponse.SC_CONTINUE);
```

The following code sample indicates that the TAI determined the user identity. WebSphere Application Server receives the user ID only and queries the user registry for additional information:

```
// modify the HttpServletResponse object
 return TAIResult.create(HttpServletResponse.SC_OK, userid);
```

The following code sample indicates that the TAI determined the user identity. WebSphere Application Server receives the complete user information that is contained in the hashtable. In this code sample, the hashtable is placed in the public credential portion of the Subject:

```
// create Subject and place Hashtable in it
 Subject subject = new Subject;
 subject.getPublicCredentials().add(hashtable);
// the response code is meaningful for only the client
 return TAIResult.create(HttpServletResponse.SC_OK, "ignored", subject);
```

The following code sample indicates that an authentication failure occured. WebSphere Application Server fails the authentication request:

```
//log error message
// ....
 throw new WebTrustAssociationFailedException("TAI failed for this reason");
```

The following methods are additional methods on the TrustAssociationInterceptor interface. These methods are used for initialization, for shutdown, and for identifying the TAI to WebSphere Application Server. For more information, see the Java documentation.

Method name

public int initialize(Properties props)

Method result

This method is called during TAI initialization and is called only if custom properties are configured for the interceptor.

Method name

public String getVersion()

Method result

This method returns the version of the TAI.

Method name
public String getType()

Method result
This method returns the type of the TAI.

Method name
public void cleanup()

Method result
This method is called when stopping the WebSphere Application Server process. Stopping the WebSphere Application Server process provides an opportunity for the TAI to perform any necessary cleanup. This method is not necessary if cleanup is not required.

Enabling a plugpoint for custom password encryption

Two properties govern the protection of passwords. By configuring these two properties, you can enable a plugpoint for custom password encryption.

Before you begin

To view an example code sample that illustrates the com.ibm.wsspi.security.crypto.CustomPasswordEncryption interface, see “Plug point for custom password encryption” on page 926.

About this task

The encryption method is called for password processing whenever the custom class is configured and custom encryption is enabled. The decryption method is called whenever the custom class is configured and the password contains the {custom:alias} tag. The custom:alias tag is stripped prior to decryption.

Procedure

1. To enable custom password encryption, you must configure two properties:
   - **com.ibm.wsspi.security.crypto.customPasswordEncryptionClass** - Defines the custom class that implements the com.ibm.wsspi.security.crypto.CustomPasswordEncryption password encryption interface.
   - **com.ibm.wsspi.security.crypto.customPasswordEncryptionEnabled** - Defines when the custom class is used for default password processing. When the passwordEncryptionEnabled option is not specified or set to false, and the passwordEncryptionClass class is specified, the decryption method is called whenever a {custom:alias} tag still exists in the configuration repository.
2. To configure custom password encryption, configure both of these properties in the server.xml file. How you perform this configuration is dependent on your existing directory structure. Choose one of the following ways to perform this configuration:
   - Place The custom encryption class (com.acme.myPasswordEncryptionClass) in a Java archive (JAR) file that resides in the ${WAS_INSTALL_ROOT}/classes directory. In this case, you have created the ${WAS_INSTALL_ROOT}/classes directory for this purpose.

     Note: WebSphere Application Server does not create the ${WAS_INSTALL_ROOT}/classes directory.
   - Place the custom encryption class (com.acme.myPasswordEncryptionClass) in a Java archive (JAR) file that resides in the ${WAS_HOME}/lib/ext directory or another valid existing directory.

   Every configuration document that contains a password (security.xml and any application bindings that contain RunAs passwords), must be saved before all of the passwords become encrypted with the custom encryption class.
3. If the custom implementation class defaults to the com.ibm.wsspi.security.crypto.CustomPasswordEncryptionImpl interface, and this class is present in

the class path, then encryption is enabled by default. This simplifies the enablement process for all nodes. It is not necessary to define any other properties except for those that the custom implementation requires. To disable encryption, but still use this class for decryption, specify the following class.

- `com.ibm.wsspi.security.crypto.customPasswordEncryptionEnabled=false`

What to do next

Whenever a custom encryption class encryption operation is called, and it creates a run-time exception or a defined PasswordEncryptException exception, the WebSphere Application Server runtime uses the {xor} algorithm to encode the password. This encoding prevents the storage of the password in plain text. After the problem with the custom class has been resolved, it automatically encrypts the password the next time the configuration document is saved.

When a RunAs role is assigned a user ID and password, it currently is encoded using the WebSphere Application Server encoding function. Therefore, after the custom plug point is configured to encrypt the passwords, it encrypts the passwords for the RunAs bindings as well. If the deployed application is moved to a cell that does not have the same encryption keys, or the custom encryption is not yet enabled, a login failure results because the password is not readable.

One of the responsibilities of the custom password encryption implementation is to manage the encryption keys. This class must decrypt any password that it encrypted. Any failure to decrypt a password renders that password to be unusable, and the password must be changed in the configuration. All encryption keys must be available for decryption there and no passwords are left using those keys. The master secret must be maintained by the custom password encryption class to protect the encryption keys.

You can manage the master secret by using a stash file for the keystore, or by using a password locator that enables the custom encryption class to locate the password so that it can be locked down.

Plug point for custom password encryption

A plug point for custom password encryption can be created to encrypt and decrypt all passwords in WebSphere Application Server that are currently encoded or decoded using Base64-encoding.

The implementation class of this plug point has the responsibility for managing keys, determining the encryption algorithm to use, and for protecting the master secret. The WebSphere Application Server runtime stores the encrypted passwords in their existing locations, preceded with {custom:alias} tags instead of {xor} tags. The custom part of the tag indicates that it is a custom algorithm. The alias part of the tag is specified by the custom implementation, which helps to indicate how the password is encrypted. The implementation can include the key alias, encryption algorithm, encryption mode, or encryption padding.

A custom provider of this plug point must implement an interface that is designed to encrypt and decrypt passwords. The interface is called by the WebSphere Application Server runtime whenever the custom plug point is enabled. The custom algorithm becomes one of the supported algorithms when the plug point is enabled. Other supported algorithms include {xor} (standard base64 encoding) and {os400} which is used on the iSeries platform.

The following example illustrates the com.ibm.wsspi.security.crypto.CustomPasswordEncryption interface:

```
package com.ibm.wsspi.security.crypto;
public interface CustomPasswordEncryption
{

    /**
     * The encrypt operation takes a UTF-8 encoded String in the form of a byte[].
     * The byte[] is generated from String.getBytes("UTF-8").
     * An encrypted byte[] is returned from the implementation in the EncryptedInfo
     * object.  Additionally, a logical key alias is returned in the EncryptedInfo
     * objectwhich is passed back into the decrypt method to determine which key was
     * used to encrypt this password. The WebSphere Application Server runtime has
```

```
     * no knowledge of the algorithm or the key used to encrypt the data.
     *
     * @param byte[]
     * @return com.ibm.wsspi.security.crypto.EncryptedInfo
     * @throws com.ibm.wsspi.security.crypto.PasswordEncryptException
     **/
    public EncryptedInfo encrypt (byte[] decrypted_bytes) throws PasswordEncryptException;

    /**
     * The decrypt operation takes the EncryptedInfo object containing a byte[]
     * and the logical key alias and converts it to the decrypted byte[].  The
     * WebSphere Application Server runtime converts the byte[] to a String
     * using new String (byte[], "UTF-8");
     *
     * @param com.ibm.wsspi.security.crypto.EncryptedInfo
     * @return byte[]
     * @throws com.ibm.wsspi.security.crypto.PasswordDecryptException
     **/
    public byte[] decrypt (EncryptedInfo info) throws PasswordDecryptException;

    /**
     * The following is reserved for future use and is currently not
     * called by the WebSphere Application Server runtime.
     *
     * @param java.util.HashMap
     **/
    public void initialize (java.util.HashMap initialization_data);
}
```

The com.ibm.wsspi.security.crypto.EncryptedInfo class contains the encrypted bytes with the user-defined alias that is associated with the encrypted bytes. This information is passed back into the encryption method to help determine how the password was originally encrypted.

```
package com.ibm.wsspi.security.crypto;
public class EncryptedInfo
{
    private byte[] bytes;
    private String alias;

/**
 * This constructor takes the encrypted bytes and a keyAlias as parameters.
 * This constructor is used to pass to or from the WebSphere Application Server
 * runtime to enable the runtime to associate the bytes with a specific key that
 * is used to encrypt the bytes.
 */

    public EncryptedInfo (byte[] encryptedBytes, String keyAlias)
    {
        bytes = encryptedBytes;
        alias = keyAlias;
    }

/**
 * This command returns the encrypted bytes.
 *
 * @return byte[]
 */
    public byte[] getEncryptedBytes()
    {
        return bytes;
    }

/**
 * This command returns the key alias.  The key alias is a logical string that is
 * associated with the encrypted password in the model. The format is
 * {custom:keyAlias}encrypted_password. Typically, just the key alias is placed
 * here, but algorithm information can also be returned.
 *
 * @return String
 */
    public String getKeyAlias()
    {
        return alias;
    }

}
```

The encryption method is called for password processing whenever the custom class is configured and custom encryption is enabled. The decryption method is called whenever the custom class is configured and the password contains the {`custom:alias`} tag . The `custom:alias` tag is stripped prior to decryption. For more information, see Enabling custom password encryption.

Implementing a custom authentication provider using JASPI

You can implement a custom authentication provider using Java Authentication SPI for Containers (JASPI, or sometimes called JASPIC) to handle the Java Platform, Enterprise Edition (Java EE) authentication of HTTP request and response messages destined for web applications.

Before you begin

For JASPI authentication processing to take place, application security must be enabled in the global or domain security configuration and the server must be restarted for the configuration changes to take effect. Read the Application security topic for more information.

About this task

This release of WebSphere Application Server supports the JSR 196: Java Authentication SPI for Containers (JASPI, or sometimes called JASPIC) specification, which enables third-party security providers to handle the Java Platform, Enterprise Edition (Java EE) authentication of HTTP request and response messages destined for web applications. The JASPI specification extends the pluggable authentication concepts of the Java Authentication and Authorization Service (JAAS) to the authentication of HTTP request and response messages. When application security is enabled, and a protected web resource is accessed, the web container and the security runtime collaborate to make an authentication decision for the caller. When using a third-party JASPI provider, the authentication decision is delegated to that provider.

The JASPI specification defines standard system programming interfaces that enable developers to write a pluggable custom authentication provider that can handle Java EE web authentication mechanisms as well as any extended authentication processing. The WebSphere Application Server runtime uses these standard system programming interfaces to invoke the JASPI authentication provider. Read the Servlet Container Profile section in the JSR 196: Java Authentication Service Provider Interface for Containers specification for the requirements that third-party authentication providers must satisfy for more information.

If application security is enabled with JASPI authentication, when the web resource (such as a servlet or a JavaServer Pages (JSP) file) is accessed, the security runtime checks if the web resource is mapped to a JASPI provider defined in the security configuration. If so, the runtime invokes the JASPI authentication provider to perform authentication for the HTTP request and response messages.

To implement a custom authentication provider using JASPI you must do the following:

Procedure

1. Develop a custom JASPI authentication provider.

   WebSphere Application Server provides support for the development of custom JASPI authentication providers to be used to perform authentication for the HTTP request and response messages destined for web applications. Read “Developing a custom JASPI authentication provider” on page 929 for more information.

2. Configure a new JASPI authentication provider.

   WebSphere Application Server allows an administrator to enable JASPI authentication and to define a third-party JASPI authentication provider as part of the global or domain security configuration. Read “Configuring a new JASPI authentication provider using the administrative console” on page 933 for more information.

3. Associate a JASPI authentication provider with an application or specific web modules.

   During application deployment, the administrator or deployer can use the Map JASPI Provider option to associate web applications and specific web modules with an existing JASPI authentication provider as defined in the security configuration. This association can also be made when editing the options for

an existing installed application. By default, an application inherits the JASPI settings defined in the WebSphere Application Server global or domain security configuration, and web modules inherit the application setting. The Map JASPI Provider option can be used to override these defaults. Read “Enabling JASPI authentication using the Map JASPI provider option during application deployment” on page 935 for more information.

Developing a custom JASPI authentication provider

You can develop a custom Java Authentication SPI for Containers (JASPI) authentication provider by creating classes that implement the required interfaces noted in the JSR 196: Java Authentication Service Provider Interface for Containers specification.

Before you begin

Review the specific interface implementation requirements for JASPI authentication providers and modules in the JSR 196: Java Authentication Service Provider Interface for Containers specification.

About this task

WebSphere Application Server supports the use of third-party authentication providers that are compliant with the servlet container profile specified in Java Authentication SPI for Containers (JASPI) Version 1.0.

The servlet container profile defines interfaces that are used by the security runtime environment in collaboration with the web container in WebSphere Application Server to invoke authentication modules before and after a web request is processed by an application. Authentication using JASPI modules is performed only when JASPI has been enabled in the security configuration and when a configured JASPI provider has been associated with the web module that processes the received web request.

To develop a custom authentication provider, create classes that implement the required interfaces noted in the JSR 196: Java Authentication Service Provider Interface for Containers specification. A provider can use one or more authentication modules for authentication. Modules can use callbacks to perform authentication, or they can manually add the necessary user identity information to the client subject. Depending on the scope of the provider, the implementation classes can be stored in various locations on the application server.

Procedure

1. Create a class that implements the javax.security.auth.message.config.AuthConfigProvider interface.

   The AuthConfigProvider implementation class must define a public two-argument constructor and the getServerAuthConfig public method:

```
import java.util.Map;
import javax.security.auth.callback.CallbackHandler;
import javax.security.auth.message.AuthException;
import javax.security.auth.message.config.AuthConfigFactory;
import javax.security.auth.message.config.AuthConfigProvider;
import javax.security.auth.message.config.ServerAuthConfig;

public class SampleAuthConfigProvider implements AuthConfigProvider {

        public SampleAuthConfigProvider(Map<String, String> properties, AuthConfigFactory factory) {
                ...
        }
        public ServerAuthConfig getServerAuthConfig(String layer, String appContext, CallbackHandler handler)
                throws AuthException {
                ...
        }
}
```

   An instance of the AuthConfigProvider implementation class is used by WebSphere Application Server when a request arrives to be processed by the web module of the application. The getServerAuthConfig method is used to obtain a ServerAuthConfig instance. The CallbackHandler argument in the method call is used by the authentication module(s).

2. Create a class that implements the javax.security.auth.message.config.ServerAuthConfig interface.

   The ServerAuthConfig implementation class must define the getAuthContextID and getAuthContext public methods:

```
import java.util.Map;
import javax.security.auth.Subject;
import javax.security.auth.message.AuthException;
import javax.security.auth.message.MessageInfo;
import javax.security.auth.message.config.ServerAuthConfig;
import javax.security.auth.message.config.ServerAuthContext;

public class SampleServerAuthConfig implements ServerAuthConfig {

        public String getAuthContextID(MessageInfo messageInfo) throws IllegalArgumentException {
                ...
        }
        public ServerAuthContext getAuthContext(String authContextID, Subject serviceSubject, Map properties)
                throws AuthException {
                ...
        }
}
```

   The getAuthContextID and getAuthContext methods in the ServerAuthConfig implementation class are used to obtain a ServerAuthContext instance.

3. Create a class that implements the javax.security.auth.message.config.ServerAuthContext interface.

   The ServerAuthContext implementation class must define the validateRequest and secureResponse public methods:

```
import javax.security.auth.Subject;
import javax.security.auth.message.AuthException;
import javax.security.auth.message.AuthStatus;
import javax.security.auth.message.MessageInfo;
import javax.security.auth.message.config.ServerAuthContext;

public class SampleServerAuthContext implements ServerAuthContext {

        public AuthStatus validateRequest(MessageInfo messageInfo, Subject clientSubject, Subject serviceSubject)
                throws AuthException {
                ...
        }
        public AuthStatus secureResponse(MessageInfo messageInfo, Subject serviceSubject)
                throws AuthException {
                ...
        }
}
```

   The validateRequest method in the ServerAuthContext implementation class is used to invoke the module that authenticates the received web request message. If the authentication result is successful, the web container dispatches the received web request message that the target web module processes in the application. If the authentication result is not successful, the request is rejected with the appropriate response status.

4. Create a class that implements the javax.security.auth.message.module.ServerAuthModule interface.

   The ServerAuthModule implementation class must define the initialize, validateRequest, and secureResponse public methods:

```
import javax.security.auth.Subject;
import javax.security.auth.callback.CallbackHandler;
import javax.security.auth.message.AuthException;
import javax.security.auth.message.AuthStatus;
import javax.security.auth.message.MessageInfo;
import javax.security.auth.message.MessagePolicy;
import javax.security.auth.message.module.ServerAuthModule;

public class SampleAuthModule implements ServerAuthModule {

       public void initialize(MessagePolicy requestPolicy, MessagePolicy responsePolicy, CallbackHandler handler, Map options)
               throws AuthException {
               ...
       }

       public AuthStatus validateRequest(MessageInfo messageInfo, Subject clientSubject, Subject serviceSubject)
               throws AuthException {
               ...
```

```
        }

        public AuthStatus secureResponse(MessageInfo messageInfo, Subject serviceSubject)
                throws AuthException {
                ...
        }
}
```

The initialize method in the ServerAuthModule implementation class is called by the ServerAuthContext implementation class to initialize the authentication module and to associate it with the ServerAuthContext instance.

The validateRequest and secureResponse methods in this class are used respectively to authenticate the javax.servlet.http.HttpServletRequest and javax.servlet.http.HttpServletResponse contained in the javax.security.auth.message.MessageInfo that is received. These methods can use the CallbackHandler instance received in the initialize method to interact with the WebSphere security runtime to validate a user password, and the active user registry to retrieve a unique-id and membership groups for a user. The retrieved data is placed in a Hashtable in the set of private credentials in the client subject. The WebSphere Application Server implementation of CallbackHandler supports three callbacks:

- CallerPrincipalCallback
- GroupPrincipalCallback
- PasswordValidationCallback

WebSphere Application Server expects the name values obtained with PasswordValidationCallback.getUsername() and CallerPrincipalCallback.getName() to be identical. If they are not, unpredictable results occur. The CallbackHandler's handle() method processes each callback given in the argument array of the method sequentially. Therefore, the name value set in the private credentials of the client subject is the one obtained from the last callback processed.

Note: Always use PasswordValidationCallback to validate a user password and to add the appropriate credentials to the client subject during authentication:

```
import javax.security.auth.Subject;
import javax.security.auth.callback.Callback;
import javax.security.auth.message.AuthException;
import javax.security.auth.message.AuthStatus;
import javax.security.auth.message.MessageInfo;
import javax.security.auth.message.callback.PasswordValidationCallback;

public AuthStatus validateRequest(MessageInfo messageInfo, Subject clientSubject, Subject serviceSubject)
        throws AuthException {
        ...
        PasswordValidationCallback pvcb = new PasswordValidationCallback(clientSubject, username, password);
        handler.handle(new Callback[] {pvcb});
        ...
}
```

If CallbackHandler is not used by the authentication module, and validateRequest returns a successful status, WebSphere Application Server requires that a Hashtable instance be included in the clientSubject with user identity information so that a custom login can be performed to obtain the credentials for the user. This Hashtable can be added to the client subject as in the following example:

```
import java.util.Hashtable;
import java.util.String;
import javax.security.auth.Subject;
import javax.security.auth.message.AuthException;
import javax.security.auth.message.AuthStatus;
import javax.security.auth.message.MessageInfo;
import com.ibm.wsspi.security.registry.RegistryHelper;
import com.ibm.wsspi.security.token.AttributeNameConstants.AttributeNameConstants;

public AuthStatus validateRequest(MessageInfo messageInfo, Subject clientSubject, Subject serviceSubject)
        throws AuthException {
        ...
        UserRegistry reg = RegistryHelper.getUserRegistry(null);
        String uniqueid = reg.getUniqueUserID(username);

        Hashtable hashtable = new Hashtable();
        hashtable.put(AttributeNameConstants.WSCREDENTIAL_UNIQUEID, uniqueid);
```

```
        hashtable.put(AttributeNameConstants.WSCREDENTIAL_SECURITYNAME, username);
        hashtable.put(AttributeNameConstants.WSCREDENTIAL_PASSWORD, password);
        hashtable.put(AttributeNameConstants.WSCREDENTIAL_GROUPS, groupList); //optional
        clientSubject.getPrivateCredentials().add(hashtable);
        ...
}
```

For more information about the Hashtable requirements and custom login, read about Developing custom login modules for a system login configuration for JAAS.

To support the login and authenticate methods of the Java Servlet 3.0 specification, the following logic must be added to the validateRequest method in the ServerAuthModule implementation class:

```
import java.util.Map;
import javax.security.auth.Subject;
import javax.security.auth.message.AuthException;
import javax.security.auth.message.AuthStatus;
import javax.security.auth.message.MessageInfo;
import javax.servlet.http.HttpServletRequest;

public AuthStatus validateRequest(MessageInfo messageInfo, Subject clientSubject, Subject serviceSubject)
        throws AuthException {
        ...
        Map msgMap = messageInfo.getMap();

        if ("login".equalsIgnoreCase(msgMap.get("com.ibm.websphere.jaspi.request"))) {
                // This request is for the login method
                String username = msgMap.get("com.ibm.websphere.jaspi.user");
                String password = msgMap.get("com.ibm.websphere.jaspi.password");
                // Authenticate using the user name and password set above.
        }
        else if ("authenticate".equalsIgnoreCase(msgMap.get("com.ibm.websphere.jaspi.request"))) {
                // this request is for the authenticate method
                String authHeader
                     = ((HttpServletRequest) messageInfo.getRequestMessage()).getHeader("Authorization");
                if (authHeader == null) {
                     // The user has not provided a username and password yet, return
                     // AuthStatus.SEND_CONTINUE to challenge
                }
                else {
                     // Authenticate using the user name and password in the authentication header.
                }
        }
        else {
                // This is not a Servlet 3.0 login or authenticate request; handle as usual.
        }
        ...
}
```

5. Compile all newly created classes.

   The following JAR files in your WebSphere Application Server installation must be specified in the class path to successfully compile the new classes:

   - *app_server_root*/dev/JavaEE/j2ee.jar
   - *app_server_root*/dev/was_public.jar (if any public WebSphere APIs were used)

6. Create a JAR file with the compiled classes.

   Depending on the requirements, the JAR file can be placed in one of three locations:

   - *app_server_root*/lib

     This location is always on the classpath for the WebSphere Application Server classloader. Using this location, the provider can be registered for a set of web modules or applications as the cell or domain default provider for all web modules and applications, and it can be registered manually as a persistent provider.

   - Shared library

     Place the provider JAR file anywhere on the WebSphere Application Server system. Configure a shared library that points to the JAR, and add that shared library to the application or server classpath. In a shared library, the provider can be registered for a set of web modules or applications, but the provider cannot be used as the cell or domain default provider. It also cannot

be registered as a persistent provider because the shared library is not in the classpath for provider registration during server startup. For more information about configuring a shared library, read about Creating shared libraries.

- Embedded in the application

  Include the provider JAR file in the application's EAR file as a utility JAR, or embed the compiled class files in the web module WAR. The embedded provider can be registered for the web modules in the application as long as the classes are included in the classpath for the web module. This provider cannot be used as a cell or domain default provider, nor can it be registered as a persistent provider. The classes in the application are not available for provider registration during server startup.

7. Configure the provider in the security configuration using the administrative console or an administration script.

   Read about “Configuring a new JASPI authentication provider using the administrative console” for more information.

Configuring a new JASPI authentication provider using the administrative console

You can configure a new Java Authentication SPI (JASPI) authentication provider in the cell or in the given security domain by using the administrative console.

About this task

This release of WebSphere Application Server supports integration of message authentication providers that are compliant with the JASPI for Containers Version 1.0 specification.

When JASPI authentication providers are configured, and WebSphere Application Server receives an HTTP request message, the security runtime environment determines if the target application is configured to use JASPI authentication. If so, the runtime environment invokes the selected authentication provider to validate the received message. Otherwise, authentication of the message request is done according to the authentication mechanism provided by WebSphere Application Server for the appropriate messaging layer.

If you want to use JASPI message authentication services, you must supply an implementation of the required interfaces as defined in the JASPI specification. Read “Developing a custom JASPI authentication provider” on page 929 for more information on these interfaces.

Authentication of HTTP request and response messages destined for JASPI-enabled deployed applications is performed according to the requirements of the Servlet Container Profile specified in the new specification.

Note: JASPI is supported in a mixed-cell environment, but can only be used in nodes that are version 8 or higher. Back-level nodes use existing authentication mechanisms.

To configure a new JASPI authentication provider using the administrative console, do the following:

Procedure

1. Click **Security > Global security**.
2. Select **Enable Java Authentication SPI (JASPI)** to enable support for JASPI authentication.
3. Click **Providers**.

   Note: It is not necessary to select **Enable Java Authentication SPI (JASPI)** until after you have configured a new JASPI authentication provider.

Note: The Default provider option is used to specify a single JASPI authentication provider to perform authentication for all web modules when JASPI authentication is enabled, and you do not override the web module to JASPI provider mapping during application deployment. During application deployment, you can override the default for every web module where it does not apply by choosing not to use JASPI or by naming a different provider to use for authentication. However, it is not recommended that you use this option unless you are certain that your default provider is capable of handling all types of web authentication (basic authentication, form authentication and client certificate authentication).

4. Click **New**.
5. Enter a name that uniquely identifies the JASPI authentication provider in the Provider name field.
6. Optional: Enter a textual description of the authentication provider in the Description field.
7. Enter the package-qualified name of the class that implements the authentication provider interface (javax.security.auth.message.config.AuthConfigProvider) in the Class name field.

   Note: In the Message layer field, WebSphere Application Server Version 8.5 supports only the HttpServlet message layer profile as defined in the JASPI specification. You cannot change this value.
8. Optional: Under Custom Properties, click **New** if you require more than one property. This parameter is a list of key/value pairs.
9. Click **OK** or **Apply**.

What to do next

You can also configure a new JASPI authentication provider by using **wsadmin** commands. Read JaspiManagement command group for the AdminTask object for more information.

Verify that your server has been restarted so that the changes to configure the JASPI provider will take effect.

Modifying an existing JASPI authentication provider using the administrative console

You can modify and configure an existing Java Authentication SPI (JASPI) authentication provider in the cell or in the given security domain by using the administrative console.

About this task

To modify and configure an existing JASPI authentication provider using the administrative console, do the following:

Procedure

1. Click **Security > Global security**.
2. Click **Providers**. You also have the option to change the Default provider from the drop-down list.

   Note: You can modify the value of the **Enable Java Authentication SPI (JASPI)** checkbox to indicate whether or not JASPI support is enabled at a later time.
3. Select an existing JASPI authentication provider to modify.
4. Optional: Enter a textual description of the authentication provider in the Description field.
5. Enter a new package-qualified name of the class that implements the authentication provider interface (javax.security.auth.message.config.AuthConfigProvider) in the Class name field if you wish to change it.

Note: In the Message layer field, WebSphere Application Server Version 8.5 supports only the HttpServlet message layer profile as defined in the JASPI specification. You cannot change this value.

6. Optional: Under Custom Properties, select an existing custom configuration property. Click **Delete** to remove the property, **Edit** to modify the property, or click **New** to create a new property. If you select **Edit** to modify an existing property, you can enter new values for the Name field and Value field if you wish to change them.
7. Click **OK** or **Apply**.

What to do next

You can also modify an existing JASPI authentication provider by using **wsadmin** commands. For more information, read JaspiManagement command group for the AdminTask object.

Verify that your server has been restarted so that the changes to configure the JASPI provider will take effect.

Deleting a JASPI authentication provider using the administrative console

You can delete an existing Java Authentication SPI (JASPI) authentication provider in the cell or in the given security domain by using the administrative console.

About this task

To delete an existing JASPI authentication provider using the administrative console, do the following:

Procedure

1. Click **Security** > **Global security**.
2. Click **Providers**, You can optionally select or deselect the **Enable Java Authentication SPI (JASPI)** check box.

   Note: You can modify the value of the **Enable Java Authentication SPI (JASPI)** checkbox to indicate whether or not JASPI support is enabled at a later time.
3. Select an existing JASPI authentication provider to delete.
4. Click **Delete**.

What to do next

You can also delete a JASPI authentication provider by using wsadmin commands. For more information, read JaspiManagement command group for the AdminTask object.

Verify that your server has been restarted so that the changes to the JASPI provider configuration will take effect.

Enabling JASPI authentication using the Map JASPI provider option during application deployment

An administrator or deployer can use the Map JASPI Provider option during application deployment to associate web applications and specific web modules with an existing Java Authentication SPI (JASPI) authentication provider as defined in the security configuration. This association can also be made when editing the options for a previously installed application.

Before you begin

Before you perform this task, verify that a JASPI authentication provider is defined as part of the global or domain security configuration. Read about “Configuring a new JASPI authentication provider using the administrative console” on page 933 for more information.

About this task

By default, an application inherits the JASPI settings defined in the WebSphere Application Server global or domain security configuration, and web modules inherit the application setting. However, you can override these default values by using the Map JASPI Provider option during application deployment. Use this option to associate a specific JASPI provider from the global or domain security configuration with the entire application or with specific web modules. You can also use this option to specify that JASPI authentication not be used for an application or specific web module.

To associate a web application or specific web modules with an existing JASPI provider:

Procedure

1. From the administrative console, click **Applications > New Application > New Enterprise Application**. Complete the required steps until you see the step for Map JASPI Provider, or click the **Map JASPI Provider** step from the installation options. A list containing the application name and associated web modules is displayed. To update a JASPI provider association after an application has been deployed, click **Applications > Application Types > WebSphere enterprise applications**, and then select the application to be modified. Click **JASPI Provider** under the Detail properties.
2. Select the application or specific web module for which the JASPI provider setting is to be modified.
3. Click the **Select JASPI Provider** menu and select one of the following options:

   Do not use JASPI
   Select to disable JASPI authentication for the selected web module or for the application.

   Inherit JASPI provider
   Select to inherit the JASPI authentication settings from default values in the cell or domain security configuration, as appropriate.

   When **Inherit JASPI provider** is selected for a web module, JASPI authentication settings for the selected module are the settings that are specified for the application.

   When **Inherit JASPI provider** is selected for the application, JASPI authentication settings are the settings that are specified in the appropriate cell or domain security configuration.

   Provider name
   When a specific provider name is selected, that provider name is used to perform authentication of web requests for the selected application or web module.
4. Complete the remaining steps to finish installing and deploying the application.

What to do next

Verify that your server has been restarted to ensure that the configuration changes to define the JASPI provider take effect. Read about “Configuring a new JASPI authentication provider using the administrative console” on page 933 for more information.

JASPI authentication providers collection

The Java Authentication Service Provider Interface (JASPI) for Containers Version 1.0 specification defines standard system programming interfaces that enable developers to write a pluggable custom authentication provider that can handle Java EE web authentication mechanisms as well as any extended authentication processing. The WebSphere Application Server runtime uses these standard system programming interfaces to invoke the JASPI authentication provider.

Read the Servlet Container Profile section in the JSR 196: Java Authentication Service Provider Interface for Containers specification for the requirements that third-party authentication providers must satisfy for more information.

If application security is enabled, and JASPI authentication is enabled with providers configured, when a web resource (such as a servlet or a JavaServer Page (JSP) file) is accessed, the security runtime checks if the web resource is mapped to a JASPI provider defined in the security configuration. If so, the runtime invokes the JASPI authentication provider to perform authentication for the HTTP request and response messages.

Note: WebSphere Application Server Version 8.5 supports only the HttpServlet message layer profile as defined in the JASPI specification.

To view this administrative console page, click **Security > Global security**. Under Authentication, click **Providers**.

To configure a new custom JASPI authentication provider in the cell or in the given security domain, click **New** and specify provider settings.

Provider name
Specifies a name that uniquely identifies the authentication provider.

Select an existing custom JASPI authentication provider name to edit and configure it.

JASPI authentication provider details
Use this page to provide configuration details for your custom Java Authentication SPI (JASPI) authentication service provider.

To view this administrative console page, click **Security > Global security**. Under Authentication, click **Providers**. Select an existing authentication service provider name or click **New** to create a new one.

Provider name
Specifies a name that uniquely identifies the authentication provider.

Description
Specifies a textual description of the authentication provider.

Class name
Specifies the package-qualified name of the class that implements the authentication provider interface (javax.security.auth.message.config.AuthConfigProvider).

Message layer
WebSphere Application Server Version 8.5 supports only the HttpServlet message layer profile as defined in the JASPI specification.

Custom properties
Specifies additional custom properties needed to initialize the authentication provider. This parameter is a list of key/value pairs.

Click **Delete** to remove a custom property or **Edit** to modify a custom property.

JASPI authentication enablement for applications
Use this page to enable or disable Java Authentication SPI (JASPI) authentication for an application or web module, and to specify the name of a JASPI authentication provider to be used for authenticating messages for the application or web module.

To view this administrative console page, click **Applications > Application Types > WebSphere enterprise applications**. Select an application, and under Detail Properties, select **JASPI provider**.

Select JASPI provider

Select to indicate the web modules in the application that you wish to specify or to override the default JASPI authentication settings for.

Select one of the JASPI provider names to choose a provider to use to perform authentication of web requests for the selected Web module or the application.

To specify how JASPI authentication is performed for the selected web module or the application, choose one of the following:

Do not use JASPI
Select to disable JASPI authentication for the selected web module or for the application.

Inherit JASPI provider
Select to inherit the JASPI authentication settings from default values in the cell or domain security configuration, as appropriate.

When Inherit JASPI provider is selected for a web module, JASPI authentication settings for the selected module are the settings that are specified for the application.

When Inherit JASPI provider is selected for the application, JASPI authentication settings are the settings that are specified in the appropriate cell or domain security configuration.

Provider name
When a specific provider name is selected, that provider is used to perform authentication of web requests for the selected application or web module.

If JASPI authentication is enabled, and a specific provider name is not specified, then the default provider name is used. For more information, read about configuring a new JASPI authentication provider using the administrative console.

If JASPI authentication is disabled, or if no default provider is selected, JASPI authentication is not performed. Web authentication is then performed according to another authentication mechanism as selected in the cell or domain security configuration.

Chapter 11. Auditing the security infrastructure

You can use the Auditing Facility to report and track auditable events to ensure the integrity of your system.

Before you begin

Before enabling the security auditing subsystem, you must enable global security in your environment.

About this task

Note: The security auditing subsystem has been introduced as part of the security infrastructure. The primary responsibility of the security infrastructure is to prevent unauthorized access and usage of resources. Utilizing security auditing has two primary goals:

- Confirming the effectiveness and integrity of the existing security configuration.
- Identifying areas where improvement to the security configuration might be needed.

Security auditing achieves these goals by providing the infrastructure that allows you to implement your code to capture and store supported auditable security events. During run time, all code other than the Java EE 5 application code is considered to be trusted. Each time a Java EE 5 application accesses a secured resource, any internal application server process with an audit point included can be recorded as an auditable event.

The security auditing subsystem has the ability to capture the following types of auditable events:

- Authentication
- Authorization
- Principal/Credential Mapping
- Audit policy management
- Delegation

Restriction: Audit instrumentation has not been included in the web services client run time.

These types of events can be recorded into audit log files. Each audit log has the option to be signed and encrypted to ensure data integrity. These audit log files can be analyzed to discover breaches over the existing security mechanisms and to discover potential weaknesses in the current security infrastructure. Security event audit records are also useful for providing evidence of accountability and nonrepudiation as well as vulnerability analysis. The security auditing configuration provides four default filters, a default audit service provider, and a default event factory. The default implementation write to a binary text-file based log. Use this topic to customize your security auditing subsystem.

Procedure

1. “Enabling the security auditing subsystem” on page 940

   Security auditing will not be performed unless the audit security subsystem has been enabled. Global security must be enabled for the security audit subsystem to function, as no security auditing occurs if global security is not also enabled.
2. Assign the auditor role to a user

   A user with the auditor role is required to enable and configure the security auditing subsystem. It is important to require strict access control for security policy management. The auditor role has been created providing granularity to allow for separation of the auditing role from the authority of the administrator. When Security Auditing is initially enabled, the cell administrator has auditor privileges. If the environment requires separation of privileges, then changes will need to be made to the default role assignments.

3. “Creating security auditing event type filters” on page 945

   You can configure event type filters to only record a specific subset of auditable event types in your audit logs. Filtering the event types that are recorded makes for easier analysis of your audit records by ensuring only those records important to your environment are archived.
4. Configuring the audit service provider.

   The audit service provider is used to format the audit data object that was passed to it before outputting the data to a repository. A default audit service provider implementation is in included. See “Configuring the default audit service providers for security auditing” on page 955 for more details on the default implementation. A third party implementation can also be coded and used. See “Configuring a third party audit service providers for security auditing” on page 959 for more details on this implementation.
5. “Configuring audit event factories for security auditing” on page 961

   The audit event factory gathers the data associated with the auditable events and creates an audit data object. The audit data object is then sent to the audit service provider to be formatted and recorded to the repository.
6. “Protecting your security audit data” on page 964

   It is important to secure and ensure the data integrity of the recorded audit data. To ensure that access to the data is restricted and tamper proof, you can encrypt and sign your audit data.
7. “Configuring security audit subsystem failure notifications” on page 953

   Notifications can be enabled to generate alerts when the security auditing subsystem experiences a failure. Notifications can be configured to record an alert in the System logs or can be configured to send an alert through email to a specified list of recipients.

Results

After successfully completing this task, you audit data will be recorded for the selected auditable events that were specified in the configuration.

What to do next

After configuring security auditing, you can analyze your audit data for potential weaknesses in the current security infrastructure and to discover security breaches that may have occurred over the existing security mechanisms. You can also use the security auditing subsystem to provide data for problem determination. If the default audit service provider was selected, the resulting binary audit log file can be read using the Audit Reader.

Enabling the security auditing subsystem

Security auditing will not be performed unless the audit security subsystem has been enabled. Global security must be enabled for the security audit subsystem to function, as no security auditing occurs if global security is not also enabled.

Before you begin

Before enabling security auditing subsystem, enable global security in your environment.

About this task

The recording of auditable security events is achieved by enabled the security auditing subsystem. Follow these steps to enable the security auditing subsystem.

Procedure

1. Click **Security > Security auditing**.

2. Select Enable security auditing. The Enable security auditing check box is not selected by default. This check box must be selected to allow security auditing to be performed with the configurations that have been specified in the `audit.xml` file.

   Note: The `audit.xml` file is used to store the audit subsystem configurations. Changes to the security auditing subsystem should be made with the user interface or the wsadmin utility. This file should not be edited manually.
3. Select the action from the Audit subsystem failure action dropdown menu to be perform when an audit subsystem failure occurs. Notifications configured to warn of a security auditing subsystem failure will not be posted if the No Warning option is selected for this field. If you select either the Log warning or the Terminate server option, then you must also configure a notification for the action to be performed.
4. Select the Auditor ID from the dropdown menu. The auditor role is needed to make changed to the security auditing configurations. By default, when auditing is first enabled, the primary administrator is also given the auditor role. The primary administrator can then add the auditor role to other users. After the auditor role is added to other users, the auditor role can be removed from the administrator to create a separation of authority between the auditor and the administrator. The Auditor ID is the user considered to be the primary auditor.
5. Optional: Select Enable verbose auditing. When an auditable event is recorded, a default set of audit data is included in the audit data object and recorded to the repository. An additional set of audit data is made available by enabling verbose auditing.
6. Click **Apply**.
7. Restart the application server. The application server must be restarted before the changes go into effect.

Results

The successful competition of these steps results in the security auditing subsystem being enabled.

What to do next

After enabling the security auditing subsystem, refinements can be made to the configuration. You might want to modify the access control of the audit subsystem to separate the authority of the administrator from the authority of the auditor. If no changes to your access control are needed, then you can configure the types of auditable security events should be recorded. To configure the types of events that are recorded, click **Event type filters**.

Security Auditing detail

The Security auditing subsystem can be enabled and configured from this panel, by users assigned the auditor role.

To view this administrative console page, click **Security > Security Auditing**. If Enable security auditing is not selected, then all of the other fields on this panel will be disabled.

Enable security auditing

The Enable security auditing check box allows users to enable or disable Security Auditing. By default, Security Auditing will not be enabled. This field corresponds with the auditEnabled field in the `audit.xml` file.

Audit subsystem failure action

The Audit subsystem failure action setting describes the behavior of the application server in the event of a failure in the auditing subsystem. Audit Notifications must be configured in order for notifications of a failure in the audit subsystem to be logged. If security auditing is not enabled, then these actions will not be performed. Failures can include an error in the interface or in the event processing. By default, the audit subsystem failure action setting is set to No warning.

The Audit subsystem failure action dropdown menu has the following options:

- No warning

  The No warning action specifies that the auditor will not be notified of a failure in the audit subsystem. The product will continue processing but audit reporting will be disabled.

- Log warning

  The Log warning action specifies that the auditor will be notified of a failure in the audit subsystem. The product will continue processing but audit reporting will be disabled.

- Terminate server

  The Terminate server action specifies the application server to gracefully quiesce when an unrecoverable error occurs in the auditing subsystem. If email notifications are configured, the auditor will be sent a notification that an error has occurred. If logging to the system log is configured, the notification of the failure will be logged to the system file.

Primary auditor user name

The Primary auditor user name dropdown menu defines a valid user which exists in the current user registry and for whom the auditor role has been given. By default, this field is blank and is a required field.

Enable verbose auditing

The Enable verbose auditing option determines the amount of audit data that is reported in an audit record. Verbose mode captures all the auditable data points, whereas not enabling verbose mode captures only a subset of the available data. This option is disabled by default.

Context object fields

Each auditable event has an associated set of information that is available for logging. This information is grouped into specific context objects. The context objects that are available for logging a specific event are specified by the event type. This topic details the information that exists for each context object and specifies whether the information is logged by default or is only logged when the verbose logging option is enabled.

The SessionContextObj object

Table 126. SessionContextObj fields. This table lists the SessionContextObj fields.

Field	Type	Description	Default or Verbose logging
sessionId	String	An identifier for the user session	Default
remoteAddr	String	The IP address for the remote host	Default
remotePort	String	The port of the remote host	Default
remoteHost	String	The host name of the remote host	Default

The PropagationContextObj object

Table 127. PropagationContextObj fields. This table lists the PropagationContextObj fields.

Field	Type	Description	Default or Verbose logging
firstCaller	String	The identity of the first user in the caller list	Default
callerList	String array	A list of names representing the identities of the users	Verbose

The RegistryContextObj object

Table 128. RegistryContextObj fields. This table lists the RegistryContextObj fields.

Field	Type	Description	Default or Verbose logging
type	String	The type of user registry being used, such as LDAP or AIX	Default

The ProcessContextObj object

Table 129. ProcessContextObj fields. This table lists the ProcessContextObj fields.

Field	Type	Description	Default or Verbose logging
domain	String	The domain to which the user belongs	Verbose
realm	String	The registry partition to which the user belongs	Default

The EventContextObj object

Table 130. EventContextObj fields. This table lists the EventContextObj fields.

Field	Type	Description	Default or Verbose logging
lastEventTrailId	String	The last ID associated with a given transaction	Verbose
eventTrailId	String array	An array of IDs that allow events that belong to a given transaction to be correlated	Default
creationTime	Date	The date an event was created	Default
globalInstanceId	Long	The unique identifier of this event	Default

The DelegationContextObj object

Table 131. DelegationContextObj fields. This table lists the DelegationContextObj fields.

Field	Type	Description	Default or Verbose logging
delegationType	String	no delegation, simple delegation, method delegation or switch user delegation	Default
roleName	String	The Run as role being used: runAsClient, runAsSpecified, runAsSystem, own ID	Default
identityName	String	Information about the mapped user	Default

The AuthnContextObj object

Table 132. AuthnContextObj fields. This table lists the AuthnContextObj fields.

Field	Type	Description	Default or Verbose logging
authnType	String	The type of authentication used	Default

The ProviderContextObj object

Table 133. ProviderContextObj fields. This table lists the ProviderContextObj fields.

Field	Type	Description	Default or Verbose logging
provider	String	The provider of the authentication or authorization service	Default
providerStatus	String	Status of whether the authentication or authorization event processed successfully by the provider	Default

The AuthnMappingContextObj object

Table 134. AuthnMappingContextObj fields. This table lists the AuthnMappingContextObj fields.

Field	Type	Description	Default or Verbose logging
mappedSecurityDomain	String	The security domain after mapping has occurred	Default
mappedRealm	String	The realm after mapping has occurred	Default

Table 134. AuthnMappingContextObj fields (continued). This table lists the AuthnMappingContextObj fields.

Field	Type	Description	Default or Verbose logging
mappedUserName	String	The user name after mapping has occurred	Default

The AuthnTermContextObj object

Table 135. AuthnTermContextObj fields. This table lists the AuthnTermContextObj fields.

Field	Type	Description	Default or Verbose logging
terminateReason	String	The reason authentication ended	Default

The AccessContextObj object

Table 136. AccessContextObj fields. This table lists the AccessContextObj fields.

Field	Type	Description	Default or Verbose logging
progName	String	The name of the program that was involved in the event	Default
action	String	The action being performed.	Default
registryUserName	String	The name of the user in the registry	Default
appUserName	String	The name of the user within an application	Default
accessDecision	String	The decision of the authorization call	Default
resourceName	String	The name of the resource in the context of the application	Default
resourceType	String	The type of resource	Default
resourceUniqueId	Long	The unique identifier of the resource	Default
permissionsChecked	String array	The permissions that were checked during the authorization call	Default
permissionsGranted	String array	The permissions that were granted during the authorization call	Default
rolesChecked	String array	The roles that were checked during the authorization call	Default
rolesGranted	String array	The roles that were granted during the authorization call	Default

The PolicyContextObj object

Table 137. PolicyContextObj fields. This table lists the PolicyContextObj fields.

Field	Type	Description	Default or Verbose logging
policyName	String	The name of the policy	Default
policyType	String	The type of policy	Default

The KeyContextObj object

Table 138. KeyContextObj fields. This table lists the KeyContextObj fields.

Field	Type	Description	Default or Verbose logging
keyLabel	String	The key or certificate label	Default
keyLocation	String	The physical location of the key database	Default
certLifetime	Date	The date when a certificate expires	Default

The CipherContextObj object

Table 139. CipherContextObj fields. This table lists the CipherContextObj fields.

Field	Type	Description	Default or Verbose logging
cipherData	Byte array	The cipher data that is captured	Verbose

The MgmtContextObj object

Table 140. MgmtContextObj fields. This table lists the MgmtContextObj fields.

Field	Type	Description	Default or Verbose logging
mgmtType	String	The type of management operation	Default
mgmtCommand	String	The application-specific command that was performed	Default
targetInfoAttributes	Target Atrribute array	Information about one or more secondary objects involved in this operation	Verbose

The ResponseContextObj object

Table 141. ResponseContextObj fields. This table lists the ResponseContextObj fields.

Field	Type	Description	Default or Verbose logging
url	String	The URL of the HTTP request	Default
httpRequestHeaders	Attributes array	The HTTP request headers provided by the client	Verbose
httpResponseHeaders	Attributes array	The HTTP response headers returned by the server	Verbose

The CustomPropertyContextObj object

Table 142. CustomPropertyContextObj fields. This table lists the CustomPropertyContextObj fields.

Field	Type	Description	Default or Verbose logging
key	String	The label representing the custom property key name	Verbose
value	Object	The object value of the custom property	Verbose

Creating security auditing event type filters

Event type filters are used to specify the types of auditable security events that are audited. Default event type filters are included with the product, but you can also configure new event type filters to specify a subset of auditable event types to be recorded by the security auditing subsystem.

Before you begin

Before configuring security auditing filters and the rest of the security auditing subsystem, enable global security in your environment. You must be assigned the auditor role to complete this task. Event type filters are used to specify what events are audited. The amount of data that is recorded for each event is specified with the **Enable verbose auditing** check box on the same panel used to enable the auditing subsystem. Navigate to **Security** > **Security auditing** to enable security auditing and determine the data recorded for each event.

About this task

Table 143. Commonly used event type filters by default in the audit.xml template file. The application server provides the following commonly used event type filters by default in the `audit.xml` template file:

Name	Event name	Outcome of event
DefaultAuditSpecification_1	SECURITY_AUTHN	SUCCESS
DefaultAuditSpecification_2	SECURITY_AUTHN	DENIED
DefaultAuditSpecification_3	SECURITY_RESOURCE_ACCESS	SUCCESS
DefaultAuditSpecification_4	SECURITY_AUTHN	REDIRECT

New event type filters can be created, or the existing default filters can be extended, to capture more event types and outcomes. Use this task to create new event type filters.

.

Procedure

1. Click **Security > Security Auditing > Event type filters> New**.
2. Enter the unique name that should be associated with this event type filter configuration in the Name field.
3. Specify the events that should be recorded when this filter is applied:
   a. Select the events that you want to be audited from the Selectable events list.
   b. Click **Add >>** to add the selected events to the Enabled events list.
   c. Select the outcomes that you want to be audited from the Selectable event outcomes list.
   d. Click **Add >>** to add the selected outcomes to the Enabled event outcomes lists.
4. Click **OK**.

Results

The successful completion of this task results in the creation of an event type filter than can be selected by the audit service providers and audit event factories to gather and record a specific set of auditable security events.

What to do next

After creating an event type filter, the filter must be specified in the audit service provider and the audit event factory to be used to gather or report audit data. The next step in configuring the security auditing subsystem is you should configure an audit service provider to define where the audit data will be archived.

Auditable security events

Auditable security events are security events that have audit instrumentation added to the security run time code to enable them to be recorded. Event filters are configured to specify which auditable security events are recorded to the audit log files.

The following list describes each valid auditable event that you can specify as an enabled event type when creating an event filter:

Table 144. Event types. Valid auditable events can be specified as an enabled event type when creating an event filter:

Event name	Description
SECURITY_AUTHN	Audits all authentication events
SECURITY_AUTHN_MAPPING	Audits events that record mapping of credentials where two user identities are involved
SECURITY_AUTHN_TERMINATE	Audits authentication termination events such as a timeout, terminated session, or user-initiated logging out

Table 144. Event types (continued). Valid auditable events can be specified as an enabled event type when creating an event filter:

Event name	Description
SECURITY_AUTHZ	Audits events related to authorization checks when the system enforces access control policies
SECURITY_RUNTIME	Audits runtime events such as the starting and the stopping of security servers. This event type is not meant for administrative operations performed by a system administrator as such operations need to use the other SECURITY_MGMT_* event types.
SECURITY_MGMT_AUDIT	Audits events that record operations related to the audit subsystem such as starting audit, stopping audit, turning audit on or off, changing configuration of audit filters or level, archiving audit data, purging audit data, and so on.
SECURITY_RESOURCE_ACCESS	Audits events that record all accesses to a resource. Examples are all accesses to a file, all HTTP requests and responses to a given web page, and all accesses to a critical database table
SECURITY_SIGNING	Audits events that record signing such as signing operations used to validate parts of a SOAP Message for web services
SECURITY_ENCRYPTION	Audits events that record encryption information such as encryption for web services
SECURITY_AUTHN_DELEGATION	Audits events that record delegation, including identity assertion, RunAs, and low assertion. Used when the client identity is propagated or when delegation involves the use of a special identity. This event type is also used when switching user identities within a given session.
SECURITY_AUTHN_CREDS_MODIFY	Audits events to modify credentials for a given user identity

For each audit event type, you must specify an outcome. Valid outcomes include SUCCESS, FAILURE, REDIRECT, ERROR, DENIED, WARNING, and INFO. Not all outcomes are applicable with all event types.

Note: Support for the SECURITY_RUNTIME auditing event type has been fully implemented for this release of WebSphere Application Server. It audits runtime events such as the starting and the stopping of security servers.

Event type filter settings

The Event type filter settings panel is used by an auditor to manage and create event type filters. Default event type filters have been included, this panel allows additional event type filters to be added. Existing event type filters are also managed using this panel.

To view this administrative console page, click one of the following paths:

- **Security > Security Auditing > Event type filters >** *event_type_filter_name*.
- **Security > Security Auditing > Event type filters > New** .

Name

The Name field specifies the unique name of the event type filter.

Enabled

The state of enablement of the filter is defined by the Enable check box. This field is represented as a boolean value. A value of true specifies that the enable field associated with the audit specification in the `audit.xml` is set to true. It does not imply that all configured event factories and service providers will be using this filter.

Filters still need to be configured for each event factory and service provider. Filters are enabled by default during configuration. However, if a filter has the enabled checkbox set to false, the filter will not gather or report data for the events and outcomes defined in that filter.

Events to associate with an audit filter

The Events to associate with an audit filter field specifies the auditable security events to be associated with this filter.

- Selectable events:

  The Selectable events list displays the available auditable security events. To enable an event for this filter, select the event from the Selectable event outcomes list and then click **Add**.
- Enabled events:

The Enabled events list displays the audit security events that are currently enabled for this filter. To disable an event for this filter, select the event from the Enabled events list and then click **Remove**.

Event outcomes to associate with an audit filter

The Event outcomes to associate with an audit filter field specifies the auditable security event outcomes to be associated with this filter.

- Selectable event outcomes:

  The Selectable event outcomes list displays the available auditable security event outcomes. To enable an event outcome for this filter, select the event outcome from the Selectable event outcomes list and then click **Add**.

- Enabled event outcomes:

  The Enabled event outcomes list displays the audit security event outcomes that are currently enabled for this filter. To disable an event outcome for this filter, select the event outcome from the Enabled event outcomes list and then click **Remove**.

Event type filters collection

The Event type filters panel displays a listing of all configured audit specifications with their unique names, the state of their enablement, and the event types and event outcomes that are specified for each configuration.

To view this administrative console page, click **Security > Security Auditing > Event type filters**.

Name

The Name field displays the unique name of the event type filter that is being represented.

Enable

The Enable check box species the state of enablement for the filter. This field is represented as a boolean value. A value of true specifies that the enable field associated with the audit specification in the `audit.xml` is set to true. It does not imply that all configured event factories and service providers will be using this filter. Filters are enabled by default when they are created. Even though it is enabled by default when it is created, the event type filter must be specified for the event factory and the audit service provider before it is actually used,

Filters still need to be configured for each event factory and service provider. A filter that is configured for an event factory or a service provider that has Enabled set to false, will not gather or report data for the events and outcomes defined in that filter.

Events and outcomes

The event types and the event outcomes that are specified by this filter. The specifications are listed in the form event_type:event_outcome and separated by commas if multiple combinations are specified by the event type filter.

Example: Generic Event Interface

This interface is used for processing generic audit events. Other interfaces can be defined which extend this interface to process specific audit event groupings, such as security events, transaction events, or other custom groupings. For WebSphere Application Server version 7.0, only security types of events are supported.

Generic Event Interface

Specific implementations might be developed to handle the data in a particular internal format. When the buildEvent() method is called, the implementation must then build the specified base event type using the internal information it has stored. After the information has been stored into a GenericEvent instance, the GenericEvent interface provides a generic way of handling the event.

```
public interface GenericEvent {

  /** * Property name used to specify the base event type to the
 * {@link GenericEvent#buildEvent} method.
 */
public static final String BASE_EVENT_TYPE = GenericEvent.class.getName() + ".baseEventType";

/**
* Returns the eventType of the event. The eventType distinguishes between these
* related events.
* The eventType depends on the particular implementation
* of the GenericEvent. For example, the Security Event implmentation has
* eventTypes such as SECURITY_AUTHN and SECURITY_AUTHZ.
* @return eventType - the eventType of the event
*/

public String getEventType();
/**
* Returns the creationTime, the creation time of the event.
* @return creationTime - the creation time of the event
*/ public Date getCreationTime(); /** * Returns the version, the version of the event.
* @return version - the version of the event
*/

public String getVersion (Properties props) throws GenericEventConfigurationException;
/**
* Returns the globalInstanceId, which is a globally unique instance
* identifier for the event.
* @return globalInstanceId - a globally unique instance identifier for the event
*/

public Long getGlobalInstanceId();
/**
* Verifies whether the event is valid; which depends on the particular
* implementation of the GenericEvent. If the event is not valid, an
* GenericEventValidationException error occurs.
*/

public void validate() throws GenericEventValidationException;
 /**
* Returns the internally wrapped base event instance after
* completing and validating the current instance of the GenericEvent.
* An GenericEvent implementation can maintain its information
* in any undisclosed internal format. The buildEvent()
* method that specifies that a specific base event type be built
* using the internal information. This allows GenericEvent implementations
* to support multiple base event formats. Thus the GenericEvent implmentation
* provides a layer of abstraction higher than the base event type.
* @param properties The value of the property BASE_EVENT_TYPE
* defines the type of the base event * @return the internally wrapped base event instance
* @throws GenericEventConfigurationException if the base event type is invalid
* or the JAR files to support that event type are not available.
* @throws GenericEventCompletionException if event completion has failed.
* @throws GenericEventValidationException if the validation has failed. This is
* validation as is performed by the validate() method.
*/

public Object buildEvent(Properties properties)
  throws GenericEventConfigurationException,
      GenericEventValidationException,
       GenericEventCompletionException;
/**
* Returns the wrapped base event instance as a string after
* completing and validating the current instance of the GenericEvent.
* An GenericEvent implementation can maintain its information
* in any undisclosed internal format. It is the buildEventString()
* method that specifies that a specific base event type be built
* using the internal information. This allows GenericEvent implementations
* to support multiple base event formats. Thus the GenericEvent implmentation
* provides a layer of abstraction higher than the base event type.
* @param properties The value of the property BASE_EVENT_TYPE
* defines the type of the base event
* @return the wrapped base event instance as a String
* @throws GenericEventConfigurationException if the base event type is invalid
* or the JAR files to support that event type are not available.
* @throws GenericEventCompletionException if event completion has failed.
* @throws GenericEventValidationException if the validation has failed. This is
* validation as is performed by the validate() method.
*/

public String buildEventString(Properties properties)
  throws GenericEventConfigurationException,
      GenericEventValidationException,
      GenericEventCompletionException;
}
```

Context objects for security auditing

Each event has an associated set of information that is available for logging. This information is grouped into specific context objects. The context objects that are available for logging a specific event are specified by the event type. All event types have the sessionContextObj, eventContextObj, accessContextObj, propogationContextObj, processContextObj and registryContextObj objects. This topic specifies which additional context objects are available for each event type.

Table 145. Context objects associated with event types. The following table describes the context objects associated with event types.

Event Type	Additional Context Objects
SECURITY_AUTHN	authnContextObj, providerContextObj
SECURITY_AUTHN_DELEGATION	delegationContextObj
SECURITY_AUTHN_MAPPING	authnMappingContextObj, providerContextObj
SECURITY_AUTHZ	providerContextObj, policyContextObj
SECURITY_ENCRYPTION	keyContextObj
SECURITY_MGMT_AUDIT	mgmtContextObj
SECURITY_RESOURCE_ACCESS	responseContextObj

For more details on the auditable data that is gather for each of these context objects, see the information for context object fields.

Context object fields

Each auditable event has an associated set of information that is available for logging. This information is grouped into specific context objects. The context objects that are available for logging a specific event are specified by the event type. This topic details the information that exists for each context object and specifies whether the information is logged by default or is only logged when the verbose logging option is enabled.

The SessionContextObj object

Table 146. SessionContextObj fields. This table lists the SessionContextObj fields.

Field	Type	Description	Default or Verbose logging
sessionId	String	An identifier for the user session	Default
remoteAddr	String	The IP address for the remote host	Default
remotePort	String	The port of the remote host	Default
remoteHost	String	The host name of the remote host	Default

The PropagationContextObj object

Table 147. PropagationContextObj fields. This table lists the PropagationContextObj fields.

Field	Type	Description	Default or Verbose logging
firstCaller	String	The identity of the first user in the caller list	Default
callerList	String array	A list of names representing the identities of the users	Verbose

The RegistryContextObj object

Table 148. RegistryContextObj fields. This table lists the RegistryContextObj fields.

Field	Type	Description	Default or Verbose logging
type	String	The type of user registry being used, such as LDAP or AIX	Default

The ProcessContextObj object

Table 149. ProcessContextObj fields. This table lists the ProcessContextObj fields.

Field	Type	Description	Default or Verbose logging
domain	String	The domain to which the user belongs	Verbose
realm	String	The registry partition to which the user belongs	Default

The EventContextObj object

Table 150. EventContextObj fields. This table lists the EventContextObj fields.

Field	Type	Description	Default or Verbose logging
lastEventTrailId	String	The last ID associated with a given transaction	Verbose
eventTrailId	String array	An array of IDs that allow events that belong to a given transaction to be correlated	Default
creationTime	Date	The date an event was created	Default
globalInstanceId	Long	The unique identifier of this event	Default

The DelegationContextObj object

Table 151. DelegationContextObj fields. This table lists the DelegationContextObj fields.

Field	Type	Description	Default or Verbose logging
delegationType	String	no delegation, simple delegation, method delegation or switch user delegation	Default
roleName	String	The Run as role being used: runAsClient, runAsSpecified, runAsSystem, own ID	Default
identityName	String	Information about the mapped user	Default

The AuthnContextObj object

Table 152. AuthnContextObj fields. This table lists the AuthnContextObj fields.

Field	Type	Description	Default or Verbose logging
authnType	String	The type of authentication used	Default

The ProviderContextObj object

Table 153. ProviderContextObj fields. This table lists the ProviderContextObj fields.

Field	Type	Description	Default or Verbose logging
provider	String	The provider of the authentication or authorization service	Default
providerStatus	String	Status of whether the authentication or authorization event processed successfully by the provider	Default

The AuthnMappingContextObj object

Table 154. AuthnMappingContextObj fields. This table lists the AuthnMappingContextObj fields.

Field	Type	Description	Default or Verbose logging
mappedSecurityDomain	String	The security domain after mapping has occurred	Default
mappedRealm	String	The realm after mapping has occurred	Default

Table 154. AuthnMappingContextObj fields (continued). This table lists the AuthnMappingContextObj fields.

Field	Type	Description	Default or Verbose logging
mappedUserName	String	The user name after mapping has occurred	Default

The AuthnTermContextObj object

Table 155. AuthnTermContextObj fields. This table lists the AuthnTermContextObj fields.

Field	Type	Description	Default or Verbose logging
terminateReason	String	The reason authentication ended	Default

The AccessContextObj object

Table 156. AccessContextObj fields. This table lists the AccessContextObj fields.

Field	Type	Description	Default or Verbose logging
progName	String	The name of the program that was involved in the event	Default
action	String	The action being performed.	Default
registryUserName	String	The name of the user in the registry	Default
appUserName	String	The name of the user within an application	Default
accessDecision	String	The decision of the authorization call	Default
resourceName	String	The name of the resource in the context of the application	Default
resourceType	String	The type of resource	Default
resourceUniqueId	Long	The unique identifier of the resource	Default
permissionsChecked	String array	The permissions that were checked during the authorization call	Default
permissionsGranted	String array	The permissions that were granted during the authorization call	Default
rolesChecked	String array	The roles that were checked during the authorization call	Default
rolesGranted	String array	The roles that were granted during the authorization call	Default

The PolicyContextObj object

Table 157. PolicyContextObj fields. This table lists the PolicyContextObj fields.

Field	Type	Description	Default or Verbose logging
policyName	String	The name of the policy	Default
policyType	String	The type of policy	Default

The KeyContextObj object

Table 158. KeyContextObj fields. This table lists the KeyContextObj fields.

Field	Type	Description	Default or Verbose logging
keyLabel	String	The key or certificate label	Default
keyLocation	String	The physical location of the key database	Default
certLifetime	Date	The date when a certificate expires	Default

The CipherContextObj object

Table 159. CipherContextObj fields. This table lists the CipherContextObj fields.

Field	Type	Description	Default or Verbose logging
cipherData	Byte array	The cipher data that is captured	Verbose

The MgmtContextObj object

Table 160. MgmtContextObj fields. This table lists the MgmtContextObj fields.

Field	Type	Description	Default or Verbose logging
mgmtType	String	The type of management operation	Default
mgmtCommand	String	The application-specific command that was performed	Default
targetInfoAttributes	Target Atrribute array	Information about one or more secondary objects involved in this operation	Verbose

The ResponseContextObj object

Table 161. ResponseContextObj fields. This table lists the ResponseContextObj fields.

Field	Type	Description	Default or Verbose logging
url	String	The URL of the HTTP request	Default
httpRequestHeaders	Attributes array	The HTTP request headers provided by the client	Verbose
httpResponseHeaders	Attributes array	The HTTP response headers returned by the server	Verbose

The CustomPropertyContextObj object

Table 162. CustomPropertyContextObj fields. This table lists the CustomPropertyContextObj fields.

Field	Type	Description	Default or Verbose logging
key	String	The label representing the custom property key name	Verbose
value	Object	The object value of the custom property	Verbose

Configuring security audit subsystem failure notifications

Notifications can be generated by a failure of the security audit subsystem. The security audit subsystem notifications can alert auditors that the security audit system is no longer recording auditable security events. Notifications are generated by a failure of the auditing subsystem, they are not related to any auditable security events or event outcome that has occurred. Notifications triggered by an event or an event outcome are not supported.

Before you begin

Before configuring notifications, enable global security and the security audit subsystem in your environment. You must be assigned the auditor role to complete this task.

About this task

If a problem is experienced with the security audit subsystem, then a notification can be generated. This is an alert that security events are no longer being audited. Notification can be written to the system log file or can be sent to a specified group of users as an email. You are able to configure notifications to alert the auditor of a problem using both of these methods simultaneously. Notifications are only generated when the Audit subsystem failure action field is set to Log warning or Terminate server.

Procedure

1. Optional: Click **Security** > **Security Auditing**.
2. Optional: Confirm the Audit subsystem failure action field is set to Log warning or Terminate server. If the Audit subsystem failure action field is set to No warning, then notifications will not be generated.
3. Click **Security** > **Security Auditing** > **Audit monitor** .
4. Under Notifications, Click **New**
5. Enter the name that should be associated with this notification configuration in the Notification name field.
6. Select the Message log check box to specify the failure notifications are recorded in the audit log.
7. Select the email sent to notification list check box to specify that failure notification email should be sent to the addresses listed in the notification list.
8. Enter an email address in the email address to add field This step is not needed if email notifications are not going to be sent.
9. Enter the mail server address in the Outgoing mail (STMP) server address. This step is not needed if email notifications are not going to be sent.
10. Click **Add >>** to add the email address and associated mail server to the email notification list.
11. Repeat steps 5 through 7 for each email address you want to specify in the email notification list.
12. Click **OK**.
13. Select the Enable monitoring check box to turn on audit failure notifications.
14. Select the notification configuration to be used from the Monitor notification dropdown menu.
15. Click **OK**.

Results

After completing this task, a notification will be generated if the security auditing subsystem experiences an unrecoverable error resulting in security events no longer being audited.

What to do next

After configuring notifications, you can analyze your audit data for potential weaknesses in the current security infrastructure and to discover possible security breaches that might have occurred.

Audit notifications cannot be removed using the administrative console. To remove an audit notification you first must run the deleteAuditNotificationMonitorByRef or the deleteAuditNotificationMonitorByName command. After running one of those commands, remove the audit notification by running the deleteAuditNotification command.

Audit monitor collection

Use this page to configure audit subsystem failure notifications. The Auditor monitor panel lists the existing notification configurations and is the gateway for creating new notification configurations and for managing the existing notification configurations.

To view this administrative console page, click **Security > Security Auditing > Audit monitor**.

Enable monitoring

Specifies whether to enable or disable notifications. If the check box is selected, then monitoring is enabled. If the check box is not selected, then monitoring is disabled. This check box is disabled by default.

Monitor notification

Specifies the notification configuration that will be used for reporting audit subsystem failures.

Notification name
Specifies a string that uniquely identifies a notification configuration.

Message log
Specifies if the configuration will send failure notifications to the message log file. If the value is true, then failure notifications will be sent to the message log file. If the value is false, then failure notifications will be not be sent to the message log file. When creating a notification, this field is in the form of a check box and is not selected by default.

Send Email
Specifies whether an email notification is sent to the addresses listed in the List of email addresses column.

List of email addresses
Specifies the email addresses listed as recipients for email notification in the event of an audit subsystem failure. No email addresses are listed by default. Email addresses will appear in this column if they are listed in the notification list in the notification, this applies even when the Email sent to notification list check box is not selected in the notification.

Audit notification settings
Use this page to create and manage notification configurations that define how auditors are made aware of audit subsystem failures.

To view this administrative console page, click **Security > Security Auditing > Audit monitor > New**.

Notification name
Specifies a string that uniquely identifies a notification configuration.

Message log
Specifies if the configuration will send failure notifications to the message log file. If the check box is selected, then failure notifications will be sent to the message log file. If the check box is not selected, then failure notifications will be not be sent to the message log file. This check box is not selected by default.

Send secure emails

Email sent to the notification list
Specifies whether the configuration will send a failure notification to the recipients listed in the notification list. If the check box is selected, then failure notifications will be sent to the recipients in the notification list. If the check box is not selected, then failure notifications will not be sent to the recipients in the notification list. This check box is not selected by default.

Email address to add
Specifies the email address to be added to the notification list to received failure notification emails. To add a recipient to the notification list, this field and the Outgoing mail (SMTP) server field must both be completed before you click the **Add**.

Outgoing mail (SMTP) server
Specifies the SMTP server to be used with this email address. If no server is specified, then the email realm will be used.

Configuring the default audit service providers for security auditing

The audit service provider is used to format the audit data object that was sent by the audit event factory. After being formatted, the audit data is recorded to the repository defined in the audit service provider configuration.

Before you begin

Before configuring the audit service provider, enable global security in your environment.

About this task

This task configures the audit service provider used to record generated audit records.

Procedure

1. Click **Security** > **Security Auditing** > **Audit service provider**.
2. Click **New** and then select **Binary file based emitter**.
3. Enter the unique name that should be associated with this audit service provider in the Name field.
4. Enter the file location of the binary log file in the Audit log file location field.

   Note: When the server is stopped, the current audit file will be saved with a timestamp in the file name; this is to facilitate archiving and to allow you to easily determine the audit files for specific periods. When you start the server again, audit data will be written to a new audit file that does not include the timestamp in the name.
5. Optional: Enter the maximum size allowed for a single binary log file in the Audit log file size field.

   This field is specified in megabytes. After the maximum audit file size is reached, a new audit file will be created or an existing audit file will be overwritten. If the maximum number of audit log files has not been set, the default maximum file value used is 10 megabytes. There is no audit archiving utility included with the product. You are responsible for the archiving of your audit data.
6. Optional: In the Maximum number of audit log files field, enter the maximum number of audit logs to be stored before the oldest is overwritten.

   The default value for this field is 100. The value of 100 is also used if the field is empty.

   Note: The maximum number of logs does not include the current binary log that is being written to. It is a reference to the maximum number of archived (timestamped) logs. The total number of binary logs that can exist for a server process is the maximum number of archived logs plus the current log.

   Also under this field, there are additional options to select the behavior when the maximum number of logs is reached. The choices are:

   oldest If you select this option, when the maximum audit logs are reached, the oldest audit log is rewritten; notification is not sent to the auditor.

   stop server
   This option does not rewrite over the oldest audit log. It stops the audit service, sends a notification to the SystemOut.log, and quiesces the application server.

   stop logging
   This option does not rewrite over the oldest audit log. It also stops the audit service, but does allow the WebSphere process to continue. Notifications are not posted in the SystemOut.log.
7. Select the filters to be used by this audit service provider. The Selectable filter list consists of a list of the configured filters that have been configured and are currently enabled.
   a. Select the filters that should be audited from the Selectable filter list.
   b. Click **Add >>** to add the selected filters to the Enabled filter list.
8. Click **Apply**.

Results

After completing these steps, your audit data will be sent to the specified repository in the format required by that repository.

What to do next

After creating an audit service provider, the audit service provider must be associated with an audit event factory provide the audit data objects to the audit service provider. Next you should configure an audit event factory.

Audit service provider collection

The Audit service provider panel displays a listing of all configured audit service provider implementations. Using this panel, a user can define a new audit service provider implementation, delete an existing implementation, and display or modify the fields associated with an existing implementation.

To view this administrative console page, click **Security > Security Auditing > Audit service provider**.

By default, the `audit.xml` will contain the IBM audit service provider implementation which emits audit records to a binary filed-based text file. This implementation is used for Binary file- based audit service provider configurations. For each existing audit service provider in the list on this panel, the unique name, type and event formatting class associated with the audit service provider will be displayed.

Name

The Name field is the unique name associated with the audit service provider implementation.

Type

The Type field specifies if the implementation is a binary file-based implementation, SMF implementation or a third party implementation.

Event formatting module class name

The event formatting class is a class used to format the generic event data object into a format that is specific to the audit service provider implementation. For example, a third party audit service provider implementation might have an event formatting class that takes the generic event and translates it into XML data. There is no Event formatting module class for binary file-based implementations nor for SMF implementations.

Audit service provider settings

Use this page to define the implementation details of the audit service provider. There are three types of audit service providers: binary file-based, third party and SMF.

To view this administrative console page, click one of the following paths:

- **Security** > **Security auditing** > **Audit service provider** > ***audit_service_provider_name***.
- **Security** > **Security auditing** > **Audit service provider** > **New** > **Binary File-based emitter**.
- **Security** > **Security auditing** > **Audit service provider** > **New** > **Third party emitter**.

Name

Specifies the unique name associated with the audit service provider.

Third party emitter class name

Specifies the name of the class for this implementation. This field is only present for Third party emitter implementations.

Audit file location

Specifies the path to the binary log file.

Audit file size

Specifies the maximum size of a single binary log file. This value is defined in megabytes.

Maximum number of audit log files
Specifies the maximum number of binary log files to create before the oldest is replaced.

Note: The maximum number of logs does not include the current binary log that is being written to. It is a reference to the maximum number of archived (timestamped) logs. The total number of binary logs that can exist for a server process is the maximum number of archived logs plus the current log.

Audit log wrapping
Specifies the wrapping behavior of the binary audit log when the maximum number of binary audit log files is reached.

There are customizable options available when specifying the default audit log wrapping behavior. This is only applicable to the Binary Audit Log implementation. Choose from one of the following options:

WRAP
If you select this option, when the maximum audit logs are reached, the oldest audit log is rewritten; notification is not sent to the auditor. This is the default option, and mimics the default behavior in WebSphere Application Server Version 7.0.

NOWRAP
This option does not rewrite over the oldest audit log. It stops the audit service, sends a notification to the `SystemOut.log`, and quiesces the application server.

SILENT_FAIL
This option does not rewrite over the oldest audit log. It also stops the audit service, but does allow the WebSphere process to continue. Notifications are not posted in the `SystemOut.log`.

Note: If audit notification of failures in the audit subsystem is configured, and SILENT_FAIL is selected, the auditor is not notified of the audit subsystem failure. The SILENT_FAIL option takes precedence

Note: If you use the NOWRAP or SILENT_FAIL options, when the server is stopped as a result of the logs being maxed-out, a stopserver is performed, or because the server abends in some way, you must archive the binary audit logs before you restart the server.

Note: This topic references one or more of the application server log files. As a recommended alternative, you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log` , `SystemErr.log`, `trace.log`, and `activity.log` files on distributed and IBM i systems. You can also use HPEL in conjunction with your native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

Event formatting module class name
Specifies a class used to format the generic event into a format that is specific to the audit service provider implementation. For example, a third party audit service provider implementation might have an event formatting class that takes the generic event and translates it into XML data.

Selectable filters
Specifies the available event filters. To enable a filter for an implementation, select the filter from the Selectable event filters list and then click **>**.

Enabled filters
Specifies the event filters that are currently enabled for an implementation. To disable a filter for an implementation, select the filter from the Enabled filters list and then click **<**.

Custom properties

Specifies any custom properties that might be used to add properties to a third party implementation. Custom properties are not available for binary file-based implementations or SMF implementations.

- Name
- Value

Example: Base Generic Emitter Interface

The Base Generic Emitter interface defines how audit events are emitted. Other interfaces can exist to extend this interface and to process specific audit events groupings, such as security events, transactional events, or some other custom grouping. Use this interface to create a custom implementation of the emitter.

Base Generic Emitter Interface

```
/**
* This is the interface for the event emitter. Event sources use this interface
* to send events to an event service.
*
*/
public interface BaseGenericEmitter {
/**
* Sends an event to the configured GenericEmitter implementation.
*
* @param event The event to be sent to the event service.
* This value cannot be null.
* @return The global instance ID of the event that was built.
* @exception GenericEmitterException If an error occurs during emitter processing.
* @exception IllegalArgumentException If the event parameter is null.
*/
public String sendEvent(GenericEvent event) throws
     GenericEventException;
/** * Sends an array of events to the configured GenericEmitter implementation.
* @param events The event array to be sent to the event service.
* This value cannot be null.
* @return The global instance IDs of the events that were built.
* @exception GenericEmitterException If an error occurs during emitter processing.
* @exception IllegalArgumentException If the events parameter is null.
*/
public String[] sendEvents(GenericEvent events[]) throws
     GenericEventException;
/**
* Causes the emitter to release all resources that are owned by this
* object and its dependents.
* Subsequent calls to this method have no effect.
*
* @throws GenericEmitterException If the emitter does release the
* held resources.
* resources.
* @throws GenericEventException If any other error occurs when releasing resources.
*/
public void close() throws
     GenericEventException;
}
```

Configuring a third party audit service providers for security auditing

The audit service provider is used to format the audit data object that was sent by the audit event factory. In addition to the default audit service provider, you may use a third party implementation as your audit service provider.

Before you begin

Before configuring the audit service provider, enable global security in your environment.

About this task

This task configures the audit service provider used to record generated audit records.

Procedure

1. Click **Security** > **Security Auditing** > **Audit service provider**.
2. Click **New** and then select **Third party emitter**.
3. Enter the unique name that should be associated with this audit service provider in the Name field.
4. Enter the Third party emitter class name.
5. Enter the Event formatting module class name. This field specifies the class used to format the generic event into a format that is specific to the audit service provider implementation. For example, your implementation might have an event formatting class that takes the generic event and translates it into XML data.
6. Select the filters to be used by this audit service provider. The Selectable filter list consists of a list of the configured filters that have been configured and are currently enabled.
   a. Select the filters that should be audited from the Selectable filter list.
   b. Click **Add >>** to add the selected filters to the Enabled filter list.
7. Optional: Enter any custom properties that you included in your third party emitter code.
8. Click **Apply**.

Results

After completing these steps, your audit data will be sent to the specified repository in the format required by that repository.

What to do next

After creating an audit service provider, the audit service provider must be associated with an audit event factory provide the audit data objects to the audit service provider. Next you should configure an audit event factory.

Example: Base Generic Emitter Interface

The Base Generic Emitter interface defines how audit events are emitted. Other interfaces can exist to extend this interface and to process specific audit events groupings, such as security events, transactional events, or some other custom grouping. Use this interface to create a custom implementation of the emitter.

Base Generic Emitter Interface

```
/**
* This is the interface for the event emitter. Event sources use this interface
* to send events to an event service.
*
*/
public interface BaseGenericEmitter {
/**
* Sends an event to the configured GenericEmitter implementation.
*
* @param event The event to be sent to the event service.
* This value cannot be null.
* @return The global instance ID of the event that was built.
* @exception GenericEmitterException If an error occurs during emitter processing.
* @exception IllegalArgumentException If the event parameter is null.
*/
public String sendEvent(GenericEvent event) throws
     GenericEventException;
/** * Sends an array of events to the configured GenericEmitter implementation.
* @param events The event array to be sent to the event service.
* This value cannot be null.
* @return The global instance IDs of the events that were built.
* @exception GenericEmitterException If an error occurs during emitter processing.
* @exception IllegalArgumentException If the events parameter is null.
*/
public String[] sendEvents(GenericEvent events[]) throws
     GenericEventException;
/**
* Causes the emitter to release all resources that are owned by this
* object and its dependents.
* Subsequent calls to this method have no effect.
```

```
*
* @throws GenericEmitterException If the emitter does release the
* held resources.
* resources.
* @throws GenericEventException If any other error occurs when releasing resources.
*/
public void close() throws
     GenericEventException;
}
```

Configuring audit event factories for security auditing

The audit event factory collects the data associated with the auditable security events and builds the audit data object. The object is then sent to the audit service provider to be formatted and recorded to a specified repository.

Before you begin

Before configuring an event factory, enable global security in your environment. An event type filter and an audit service provider need to be created before completing these steps

About this task

Procedure

1. Click **Security** > **Security Auditing** > **Audit event factory configurations** > **New**.
2. Enter the unique name that should be associated with this Audit event factory configuration in the Name field.
3. Select either **IBM audit event factory** or **Third party event factory**.
   a. Enter the Third party audit event factory class name. This step is only required if a Third party event factory is being created.
4. Select the appropriate audit service provider implementation from the Audit service provider dropdown menu,
5. Select the event type filter configuration to be used by this audit event factory. The Filters list consists of a list of the event type filter configurations that have been created and are currently enabled.
   a. Select the event type filters that should be used from the Selectable filter list.
   b. Click **Add >>** to add the selected event type filter configurations to the Enabled filter lists.
6. Enter any Custom properties that need to be included with this audit event factory configuration. Custom properties are only available for Third party event factory implementations.
7. Click **Apply**.

Results

After successful completion of these steps, you will have an event factory that can be used to gather auditable event data.

What to do next

After configuring an audit event factory, you can optionally protect your data by configuring the security auditing subsystem to sign and encrypt your audit logs.

Audit event factory configuration collection

The Audit event factory configuration panel displays a list of all currently configured audit event factory implementations. This panel allows a user with the auditor role to manage their configured audit event factories. This includes the ability to configure a new implementation, which is done using the **New** button on this panel.

To view this administrative console page, click **Security > Security Auditing > Audit event factory configuration**.

Name
The Name field specifies the unique name associated with the audit event factory configuration.

Type
The Type field specifies this audit event factory configuration as either an IBM audit event factory or a Third party audit event factory.

Class name
The Class name field specifies the class that is being implemented in an audit event factory configuration.

The class name is `com.ibm.ws.security.audit.AuditEventFactoryImpl` for an IBM event factory. For a Third party audit event factory, the class name is the class specified in the Third party audit event factory class name field.

Audit event factory settings
The Audit event factory settings panel displays the details of a specific audit event factory. The auditor uses this panel to manage and create audit event factory configurations.

To view this administrative console page, click on of the following paths:

- **Security** > **Security Auditing** > **Audit event factory configuration** > ***audit_event_factory_configuration_name***.
- **Security** > **Security Auditing** > **Audit event factory configuration** > **New**.

Name
Specifies the unique name associated with the audit event factory configuration.

Type
Specifies this audit event factory configuration as either an IBM audit event factory or a Third party audit event factory. This field does not appear on the panel during the creation of a new audit event factory. It is included when viewing or modifying an existing audit event factory.

IBM audit event factory
Specifies that the Type field of this audit event factory is IBM audit event factory. This check box only appears on the panel during the creation of a new audit event factory. This check box is selected by default when creating a new audit event factory.

Third party audit event factory
Specifies that the Type field of this audit event factory is Third party audit event factory. This check box only appears on the panel during the creation of a new audit event factory. This check box is not selected by default when creating a new audit event factory.

- The Third party audit event factory class name field is active when the Third party audit event factory check box is selected. This field represents the class name of the third-party implementation of the Audit Event Factory interface

Class name
Specifies the class that is being implemented in a audit event factory configuration.

Although not specified during creation, the class name is `com.ibm.ws.security.audit.AuditEventFactoryImp` for an IBM event factory.

Audit service provider
Specifies where the audit data objects gathered by this audit event factory will be sent.

Selectable filters
Specifies the filters that are currently available to be used for an implementation. To enable a filter for an implementation, select the filter from the Selectable filter list and then click **>**.

Enabled filters
Specifies the filters that are currently enabled for an implementation. To disable a filter for an implementation, select the filter from the Enabled filter list and then click **<**.

Custom properties
Specifies properties that the auditor can define to configure the Audit Event Factory implementation. This might be used by third party implementation of the audit event factory interface. Custom properties are not used for the IBM audit event factory implementation.

Each custom property has the following fields:

- Name
- Value

Example: Generic Event Factory Interface
This interface is used for processing generic audit events. Other interfaces can be defined which extend this interface to process specific audit event groupings, such as security events, transaction events, or some other custom grouping.

Generic Event Factory Interface
```
/**
* GenericEventFactory is the interface that is used to generate audit events.
* This interface may be extended to generate application specific audit events.
*
* One or more GenericEventFactory implementations each with a unique name can be defined in the
* security configuration and be used by WebSphere Application Server security auditing service.
* @author IBM Corporation
* @version WAS 7.0
* @since WAS 7.0
*/
public interface GenericEventFactory {
/**
* The init method allows a GenericEventFactory implementation to
* initialize its internal auditing configuration using the properties and context object.
*
* The properties and context objects are treated as read-only and must not be modified by the
* GenericEventFactory implementation.
*
* @param A String object represents the name of this GenericEventFactory.
* @param A Map properties object that contains the custom properties that can be defined in the
* the admin console or by using wsadmin scripting tool.
* @param A Map object that contains the context that includes cell name, node name, and server name.
* @exception ProviderFailureException might occur if the audit factory does not initialize
*/
public void init(String name, Map properties, Map context) throws ProviderFailureException;
/**
*
* The terminate method gracefully quiesces the event factory implementation.
*/
public void terminate();
/**
*
* The refresh method allows a GenericEventFactory implementation to
* update its internal auditing configuration using the properties object.
*
* The properties object is treated as read-only and must not be modified by the
* GenericEventFactory implementation.
*
* @param A Map object that contains the custom properties
* @exception ProviderFailureException might occur if the factory does not refresh
*/
public void refresh(java.util.Map properties) throws ProviderFailureException;
/**
*
* The getName method returns the name of this GenericEventFactory.
*
* @param None
* @return a String object represents the name of the GenericEventFactory.
*/
public String getName();
/**
```

```
*
* The sendEvent method determines whether the specified audit event is generated by this
* GenericEventFactory.
*
* @param a String object represents an audit event
* @param a OutcomeType object represents the audit outcome value
* @exception ProviderFailureException might occur if the audit factory does not initialize
* @return a boolean success/failure
* @exception ProviderFailureException might occur if the audit factory does not send the event.
*/
public boolean sendEvent(String auditEventType, OutcomeType auditOutcome) throws
ProviderFailureException;
*
}
```

Protecting your security audit data

The security auditing subsystem allows for protection of your security audit data by increasing the assurance that the audit data has not been tampered or modified outside of the auditing facility. This option also protects the confidentiality of the data. The audit data is protected by encrypting and signing the recording data.

Before you begin

Restriction: Signing and encrypting your audit data is only available for data created using the default binary log audit service provider. If you are using the SMF emitter or a 3rd party emitter you will not be able to sign or encrypt your data.

Before configuring protection for your security audit data, enable global security and security auditing in your environment. You must be assigned the auditor role to complete the task of protecting your audit data. You will also need the administrator role to configure your audit data to be signed.

About this task

The practice of auditing requires assurances that your audit data is accurate and uncompromised. Your audit data has the option to be encrypted, signed, or encrypted and signed. You can protect your audit data using these options to provide assurances that you data is only viewed by authorized users and can not untraceably be modified . To protect the validity of your security auditing functionality, complete the following steps:

Procedure

1. “Encrypting your security audit records” on page 965 Audit logs can be encrypted to ensure your audit data is protected. The audit logs will be encrypted using a certificate that is saved to a keystore in the `audit.xml` file. By encrypting your audit records, only users with the password to the keystore will be able to view or update the audit logs.
2. “Signing your security audit records” on page 966 Audit logs can be signed to ensure the integrity of your audit data. By signing your audit records, you ensure any modifications of the audit logs can be traced.

Results

After completing these steps your data will be signed, encrypted or signed and encrypted to provide assurances that the data is accurate and confidential.

What to do next

After protecting your data, you can configure notifications to ensure you are notified if a problem with the security auditing subsystems occurs that prevents security events from being recorded.

Encrypting your security audit records

Audit logs can be encrypted to ensure your audit data is protected. By encrypting your audit records, only users with access to the encrypting certificate will be able to view the audit logs.

Before you begin

Restriction: Encrypting audit data is only available for data created using the default audit service provider. If you are using the SMF emitter or a 3rd party emitter you will not be able to encrypt your data.

Before configuring your security audit records to be encrypted, enable global security and security auditing in your environment. You must be assigned the auditor role to encrypt your security auditing records. If you are using a certificate stored in the `security.xml` file, you also require the administrator role to complete this task.

About this task

Procedure

1. Click **Security** > **Security Auditing** > **Audit record encryption configuration**.
2. Select the Enable encryption check box to specify that your audit records should be encrypted. All other fields on this panel will be unavailable until this check box has been selected.
3. Select the keystore that contains the encrypting certificate from the dropdown menu or click **New** to create a new certificate in an existing keystore. Use the following steps if you are creating a new certificate:
   a. Enter the name of the keystore in the Name field.
   b. Enter the path to the keystore file in the Path field.
   c. Enter the password to be associated with the keystore in the Password field.
   d. Confirm the password associated with the keystore by retyping the password in the Confirm password field.
   e. Select the keystore type from the Type dropdown list. The default value of the Type dropdown list is PKCS12.
4. If you are using an existing certificate to encrypt your audit records, ensure **Certificate in keystore** is selected and specify the intended certificate in the Certificate alias dropdown menu.
5. If you are generating a new certificate to encrypt your audit records, select **Create a new certificate in the selected keystore** and follow these steps:
   a. Enter the name of your new certificate in the Certificate alias field.
   b. Select either Automatically generate certificate or Import a certificate. The certificate used to encrypt the data in the audit log files can either be created or imported. If you selected to generate a certificate, then skip to the last step on this page. If you selected to import a certificate, then continue on with step c.
   c. Enter the name of the keystore file in the Key file name field.
   d. Enter the path to the keystore file in the Path field.
   e. Select the keystore type from the Type dropdown list. The default value of the Type dropdown list is PKCS12.
   f. Enter the password associated with the keystore in the Key File password field.
   g. Click **Get key file aliases** to populate the Certificate alias to import dropdown menu.
   h. Select the certificate to be imported from the Certificate alias to import dropdown menu.
6. Click **OK**.

Results

After completing these steps, your audit logs will be encrypted to ensure only authorized users can view the content of your audit log files.

What to do next

After you have finished configuring your audit logs to be encrypted, you can ensure the data integrity of your audit logs by configuring the audit subsystem to sign your audit records.

Signing your security audit records

Audit logs can be signed to ensure the integrity of your audit data. By signing your audit records, modifications of the audit logs can be traced.

Before you begin

Restriction: Signing audit data is only available for data created using the default audit service provider. If you are using the SMF emitter or a 3rd party emitter you will not be able to sign your data.

Before configuring your security audit records to be signed, enable global security and security auditing in your environment. You must be assigned the auditor role and the administrator role to configure audit record signing.

About this task

Procedure

1. Click **Security** > **Security Auditing** > **Audit record signing configuration**.
2. Select the Enable signing check box to specify that your audit records should be signed. All other fields on this panel will be unavailable until this check box has been selected.
3. Select the keystore that contains the signing certificate from the Managed keystore containing the signing certificate dropdown menu.
4. If you are using an existing certificate to sign your audit records, ensure Certificate in keystore is selected and specify the intended certificate in the Certificate alias dropdown menu.
5. If you are generating a new certificate to sign your audit records, select Create a new certificate in the selected keystore and follow these steps:
   a. Enter the name of your new certificate in the Certificate alias field.
   b. Select on of the following options: Import the encryption certificate, Automatically generate certificate or Import a certificate. The certificate used to encrypt the data in the audit log files can either be created or imported.
      - If you selected Import the encryption certificate, then you will use the encryption certificate to also sign your audit records. Skip to the last step on this page to complete this configuration.
      - If you selected to generate a certificate, then skip to the last step on this page to complete this configuration.
      - If you selected to import a certificate from an existing keystore, then continue on with step c.
   c. Enter the name of the keystore file in the Key file name field.
   d. Enter the path to the keystore file in the Path field.
   e. Select the keystore type from the Type dropdown list. The default value of the Type dropdown list is PKCS12.
   f. Enter the password associated with the keystore in the Key File password field.
   g. Click **Get key file aliases** to populate the Certificate alias to import dropdown menu.
   h. Select the certificate to be imported from the Certificate alias to import dropdown menu.

6. Click **OK**.

Results

After you have completed these steps, your audit logs will be digitally signed to ensure the integrity of the data.

What to do next

After you have finished configuring your audit logs to be signed, you can ensure the confidentiality of your audit logs by configuring the audit subsystem to encrypt your audit records.

Audit encryption keystores and certificates collection

The Audit encryption keystores and certificates panel allows the auditor to manage the keystores and certificates used for audit encryption.

To view this administrative console page, click **Security > Security Auditing > Audit encryption keystores and certificates**.

Name

Specifies the unique name of the keystores used for storing the encryption certificate.

Path

Specifies the path to the listed keystore file.

The path to the keystore file can be listed using environment variables, ${PROFILE_ROOT}, or with a fully qualified path.

Audit record encryption configuration settings

Use this page to enable encryption for your audit records. Encrypting your audit records ensures only a user given access to the certificate used for encryption is allowed to view the audit records.

To view this administrative console page, click **Security > Security auditing > Audit record encryption configuration**. If **Enable encryption** is not selected, then all of the other fields on this panel will be disabled. Encryption is not enabled by default.

Enable encryption

Specifies whether your audit records will be encrypted. This check box is not selected by default.

Audit keystore containing the encryption certificate

Specifies the audit keystore specified to store the encryption certificate.

A new keystore can be created by clicking on the **New...** button.

Certificate in keystore

Specifies an existing certificate will be used from the keystore specified in the Audit keystore containing the encryption certificate field. This field is selected by default. If a keystore in the `security.xml` file is used, the administrator role is required.

- Certificate alias

  When the **Certificate in keystore** field is selected, the certificate alias dropdown menu displays a list of certificate aliases contained in the keystore defined by the **Audit keystore containing the encryption certificate** field. Select the certificate from the dropdown menu to be used to encrypt your audit records.

Create a new certificate in the selected keystore
Specifies that a new certificate will be created in the keystore defined by the **Audit keystore containing the encryption certificate** field.

- Certificate alias

  When the **Create a new certificate in the selected keystore** is selected, the **Certificate alias** field is used to define the name of the certificate to be created in the keystore defined by the **Audit keystore containing the encryption certificate** field.
- Automatically generate certificate

  When selected, the **Automatically generate certificate** field specifies that the application server will automatically generate the certificate. This field is selected by default when the **Create a new certificate in the selected keystore** field is selected.
- Import a certificate

  When selected, the **Import a certificate** field specifies that an existing self-signed certificate will be imported by the auditor into the keystore and used to encrypt your audit records. This field is not selected by default when the **Create a new certificate in the selected keystore** field is selected. The following fields need to be defined to import an existing certificate.
  - The **Key file name** field specifies the keystore filename that contains the certificate to be imported.
  - The **Path** field specifies the path to the keystore file that contains the certificate to be imported.
  - The **Type** field specifies the type of the keystore file that contains the certificate to be imported.
  - The **Key file password** field specifies the password used to access the keystore file that contains the certificate to be imported.
  - **Certificate alias to import** field specifies the alias of the certificate to be imported.

Audit record signing configuration settings
Use this page to enable signing for your audit records. Signing audit records ensures tamper-proof recording of the auditable events. Both the auditor and administrator roles are required to configure the signing of your audit data.

To view this administrative console page, click **Security** > **Security auditing** > **Audit record signing configuration**. If **Enable signing** is not selected, then all of the other fields on this panel will be disabled.

Enable signing
Specifies whether your audit records will be encrypted. This check box is not selected by default.

Managed keystore containing the signing certificate
Specifies the keystore used to store the signing certificate.

Certificate in keystore
Specifies an existing certificate will be used from the keystore specified in the **Managed keystore containing the signing certificate** field. This field is selected by default.

- Certificate alias

  When the **Certificate in keystore** field is selected, the **Certificate alias** dropdown menu displays a list of certificate aliases contained in the keystore defined by the **Managed keystore containing the signing certificate** field. Select the certificate from the dropdown menu to be used to sign your audit records.

Create a new certificate in the selected keystore
Specifies that a new certificate will be created in the keystore defined by the **Managed keystore containing the signing certificate** field.

- Certificate alias

When the **Create a new certificate in the selected keystore** is selected, the **Certificate alias** field is used to define the name of the certificate to be created in the keystore defined by the **Audit keystore containing the encryption certificate** field.

- Import the encryption certificate

  Specifies the certificate used for encryption will be imported into the signing keystore file and used for signing.

- Automatically generate certificate

  Specifies the application server will automatically generate the certificate. This field is selected by default when the **Create a new certificate in the selected keystore** field is selected.

- Import a certificate

  Specifies an existing self-signed certificate will be imported by the auditor into the keystore and used to encrypt your audit records. This field is not selected by default when the **Create a new certificate in the selected keystore** field is selected. The following fields need to be defined to import an existing certificate.

  – The **Key file name** field specifies the keystore filename that contains the certificate to be imported.
  – The **Path** field specifies the path to the keystore file that contains the certificate to be imported.
  – The **Type** field specifies the type of the keystore file that contains the certificate to be imported.
  – The **Key file password** field specifies the password used to access the keystore file that contains the certificate to be imported.
  – **Certificate alias to import** field specifies the alias of the certificate to be imported.

Audit record keystore settings

The Audit record keystore panel is used by an auditor to define the keystores used for storing the encryption certificate used to encrypt the audit records. Keystores used for auditing are managed outside of other keystores being used on the system to facilitate separation of the authority of the auditor for the authority of the administrator.

To view this administrative console page, click one of the following paths:

- **Security > Security Auditing > Audit encryption keystores and certificates >** *keystore_name*.
- **Security > Security Auditing > Audit encryption keystores and certificates > New**.
- **Security > Security Auditing > Audit record encryption configuration > New**

Name

The Name field specifies the unique name for the keystore. This is a required field.

Path

Specifies the path where the keystore file is located. This is a required field.

Password

Specifies the password to be used for this keystore. This is a required field.

Confirm Password

Specifies confirmation of the value provided in the Password field. This is a required field.

Type

The Type field specifies the type of the keystore. The Type dropdown menu has the following options for defining the keystore type:

- JCEKS
- CMSKS
- PKCS12 - The default value for the Type field is PKCS12.
- Cryptographic Token Device (PKCS11)

- JKS
- PKCS12JarSigner

Using the audit reader

The audit reader is a utility that can be used to read the binary audit logs generated by the default binary emitter implementation. The audit reader parses the audit log to generate an HTML report. The audit reader is invoked using wsadmin commands and is not accessible using the administrative console.

Before you begin

The audit reader can only be used to parse log files that are created by the default audit service provider. Logs created by a third-party emitter can not be parsed by the audit reader.

About this task

Your audit logs might be encrypted, signed, encrypted and signed or neither encrypted nor signed. The audit reader is able to parse any of these combinations to generate an HTML report. If the audit log file is encrypted, the password of the keystore storing the certificate used to encrypt the log must be provided. The showAuditLogEncryptionInfo wsadmin command can be used to get information to determine which keystore was used to sign the audit log.

Depending on the selections you made in your audit service provider configuration, the size of the audit logs can become large enough to make them cumbersome to review. What data has been recorded into your log is dependant on the event type filers you are using and whether you specified to use verbose logging. Options are provided for you to further limit the data included in the HTML report that is generated by the audit reader to a subset that you specify. The audit reader can be used to parse the same data multiple times to generate separate reports for your different requirements.

By default, all event types, outcome types, timestamps, and sequence numbers will be gathered from the Binary Audit log and generated into a report. The ability to specify only specific event types, only specific sequence numbers, only records with specific timestamps, as well as specific outcome types is provided. A sequence number is a unique identifier assigned to each audit record. Options exist to limit which events, outcomes, and sequence numbers are included in the report.

The report type controls what data is reported for each audit record in the log file. The default report type includes the follow data for each audit record:

- creationTime
- action
- progName
- registryType
- domain
- realm
- remoteAddr
- remotePort
- remoteHost
- resourceName
- resourceType
- resourceUniqueId

The complete report type generates a report based on all the data that was logged for the selected audit records. The complete report type includes all the data that is included by the default report type and all

the additional datapoints that were logged for these audit records. The additional available datapoints for an audit record varies depending on the event type it represents.

A custom report type is also included. Use the custom report type to specify only the datapoints that you want generated in the report. A report may be generated based on the following criteria:

- all or specific event types
- all or specific outcome types
- all or a specific sequence number range
- all or a specific timestamp range

Procedure

Run the binaryAuditLogReader wsadmin command to use the audit reader to generate a log report. See the AuditReaderCommands command group for the AdminTask object article for more information.

Results

After you complete these steps, you will generated an HTML report containing the data specific to your requirement.

Example

Audit Event Outcome Codes

In a binary audit log or the output of the audit reader tool, audit event outcomes are expressed with a numeric code. Use this table to associate the audit event outcome code in the binary audit logs to a generic error messages.

Table 163. Event Outcome Codes. This table lists the event outcome codes.

Outcome reason code	Description
0	An error occurred while parsing the certificate.
1	The security context does not exist for the thread.
2	There is conflicting session evidence.
3	The session has been rejected.
4	The token has expired.
5	Successful authentication has occurred.
6	Successful authentication for accessing a resource has occurred.
7	Successful authentication occurred while mapping a user.
8	Successful authorization has occurred.
9	Login termination was successful.
10	Invalid evidence exists.
11	There was a GSS formatting error.
12	Credentials were unauthenticated.
13	Authentication failed.
14	An invalid resource was accessed.
15	Authentication was denied.
16	Authorization was denied.
17	Access was denied because of an authentication failure.
18	Authorization was excluded.
19	Authorization was excluded because of access without proper security role.
20	An unsupported authentication mechanism was used.
21	An authentication redirect occurred.
22	The context does not exist.

Table 163. Event Outcome Codes (continued). This table lists the event outcome codes.

Outcome reason code	Description
23	A TAI challenge occurred.
24	A TAI validation was not successful.
25	A TAI mapping was not successful.
26	A provider failure occurred.
27	A SSO token validation was not successful.
28	An invalid user id or password was provided.
29	A send login form
30	An invalid configuration exists.
31	An user id or password is missing.
32	Failure occurred for an unknown reason.
33	The account was disabled because of retry violations.
34	The account was locked out because of retry violations.
35	The account was locked out because the maximum number of unsuccessful login attempts has occurred.
36	The account is disabled.
37	The account has expired.
38	The account is unlocked.
39	The maximum inactive time permitted for the account has elapsed.
40	The password has expired.
41	The minimum interval for a password change has unexpired.
42	The maximum interval permitted before a password must be changes has elapsed.
43	An authentication failure has occurred.
44	An invalid user name was provided.
45	A pin is required.
46	This outcome code is not used in this release.
47	A user mapping did not occur successfully.
48	A certificate failure occurred.
49	A policy violation has occurred.
50	A policy violation has occurred because of the time of day.
51	The policy allows access.
52	A policy violation has occurred because the maximum number of unsuccessful login attempts has been reached.
53	A user name mismatch has occurred.
54	An invalid user password was provided.
55	A token signature violation has occurred.
56	The token is not yet valid.
57	The token is not supported.
58	The token is not in a valid format.
59	A credential mapping failure occurred.
60	The delegate is not authorized.
61	Access to a resource is unauthorized because of an authorization.
62	Access to a resource is unauthorized because of a time of day policy.
63	Access to a resource is unauthorized.
64	Access to a resource is unauthorized because of quality of protection.
65	Access to a resource is unauthorized because of an authorization level.
66	Access to a resource is unauthorized because reauthentication is required.
67	A password error has occurred because it does not meet password standards: minimum alphabetic characters required.
68	A password error has occurred because it does not meet password standards: minimum alphanumeric characters required.
69	A password error has occurred because it does not meet password standards: minimum numeric characters required.

Table 163. Event Outcome Codes (continued). This table lists the event outcome codes.

Outcome reason code	Description
70	A password error has occurred because it does not meet password standards: minimum alphabetic low case characters required.
71	A password error has occurred because it does not meet password standards: minimum alphabetic upper case characters required.
72	A password error has occurred because it does not meet password standards: minimum special characters required.
73	A password error has occurred because it does not meet password standards: maximum repeated characters exceeded.
74	A password error has occurred because it does not meet password standards: contains user name
75	A password error has occurred because it does not meet password standards: reused password.
76	A password error has occurred because it does not meet password standards: contains previous password.
77	A password error has occurred because it does not meet password standards: violations in number of characters.
78	A password error has occurred because it does not meet password standards: first or last characters are numeric.
79	An illegal form login configuration exists.
80	Access is denied because of a incorrect URI.
81	Start was successful
82	Stop was successful.
83	The audit subsystem has been stopped.
84	The audit subsystem has successfully been enabled.
85	The audit subsystem has had a successful policy change.
86	Delegation was successful.
87	Delegation was not successful.
88	The audit subsystem has successfully been disabled.
89	An audit subsystem has occurred because a security header is missing.
90	An audit timestamp has been confirmed.
91	A bad audit timestamp has occurred.
92	Audit confidentially has been confirmed
93	Audit confidentially cannot be confirmed.
94	An audit decryption error has occurred.
103	A login attempt has been made by a user who has already logged in successfully.

Chapter 12. Tuning, hardening, and maintaining security configurations

After installing WebSphere Application Server, there are several considerations for tuning, strengthening, and maintaining your security configuration.

About this task

The following topics are covered in this section:

Procedure

- Tuning security configurations You can tune your security configuration to balance performance with function. You can achieve this balance following considerations for tuning general security, Common Secure Interoperability version 2 (CSIv2), Lightweight Directory Access Protocol (LDAP) authentication, web authentication, and authorization. For more information on tuning security, see "Tuning security configurations."
- Hardening security configurations Several methods exist that you can use to protect your infrastructure and applications from different forms of attack. For more information on hardening your security, see "Hardening security configurations" on page 980.
- Securing passwords in files Password encryption and encoding can add protection to passwords existing in files. For more information on encoding and encrypting passwords, see "Securing passwords in files" on page 983.

What to do next

For additional information about hardening security configurations, see the WebSphere Application Server security web page.

Tuning security configurations

You can tune security to balance performance with function. You can achieve this balance following considerations for tuning general security, Common Secure Interoperability version 2 (CSIv2), Lightweight Directory Access Protocol (LDAP) authentication, web authentication, and authorization.

About this task

Performance issues typically involve trade-offs between function and speed. Usually, the more function and the more processing that are involved, the slower the performance. Consider what type of security is necessary and what you can disable in your environment. For example, if your application servers are running in a Virtual Private Network (VPN), consider whether you can disable Secure Sockets Layer (SSL). If you have a lot of users, can they be mapped to groups and then associated to your Java Platform, Enterprise Edition (Java EE) roles? These questions are things to consider when designing your security infrastructure.

Procedure

- Consider the following recommendations for tuning general security.
  - Consider disabling Java 2 security manager if you know exactly what code is put onto your server and you do not need to protect process resources. Remember that in doing so, you put your local resources at some risk.
  - Consider increasing the cache and token timeout if you feel your environment is secure enough. By increasing these values, you have to re-authenticate less often. This action supports subsequent requests to reuse the credentials that already are created. The downside of increasing the token timeout is the exposure of having a token hacked and providing the hacker more time to hack into

the system before the token expires. You can use security cache properties to determine the initial size of the primary and secondary hashtable caches, which affect the frequency of rehashing and the distribution of the hash algorithms.

See the article “Authentication cache settings” on page 157 for a list of these properties.

- Consider changing your administrative connector from Simple Object Access Protocol (SOAP) to Remote Method Invocation (RMI) because RMI uses stateful connections while SOAP is completely stateless. Run a benchmark to determine if the performance is improved in your environment.
- Use the wsadmin script to complete the access IDs for all the users and groups to speed up the application startup. Complete this action if applications contain many users or groups, or if applications are stopped and started frequently. WebSphere Application Server maps user and group names to unique access IDs in the authorization table. The exact format of the access ID depends on the repository. The access ID can only be determined during and after application deployment. Authorization tables created during assembly time do not have the proper access IDs. See the Commands for the AdminApp article for more information about how to update access IDs.
- Consider tuning the Object Request Broker (ORB) because it is a factor in enterprise bean performance with or without security enabled. Refer to the information about ORB tuning guidelines.
- If using SSL, enable the SSL session tracking mechanism option as described in the information about session management settings.
- In some cases, using the unrestricted Java Cryptography Extension (JCE) policy file can improve performance. Refer to the information about tuning Web Services Security.
- Distributing the workload to multiple Java virtual machines (JVMs) instead of a single JVM on a single machine can improve the security performance because there is less contention for authorization decisions.

- Consider the following steps to tune Common Secure Interoperability version 2 (CSIv2).
  - Consider using Secure Sockets Layer (SSL) client certificates instead of a user ID and password to authenticate Java clients. Because you are already making the SSL connection, using mutual authentication adds little overhead while it removes the service context that contains the user ID and password completely.
  - If you send a large amount of data that is not very security sensitive, reduce the strength of your ciphers. The more data you have to bulk encrypt and the stronger the cipher, the longer this action takes. If the data is not sensitive, do not waste your processing with 128-bit ciphers.
  - Consider putting only an asterisk (*) in the trusted server ID list (meaning trust all servers) when you use identity assertion for downstream delegation. Use SSL mutual authentication between servers to provide this trust. Adding this extra step in the SSL handshake performs better than having to fully authenticate the upstream server and check the trusted list. When an asterisk (*) is used, the identity token is trusted. The SSL connection trusts the server through client certificate authentication.
  - Ensure that stateful sessions are enabled for CSIv2. This is the default, but requires authentication only on the first request and on any subsequent token expirations.
  - Consider changing the values for the CSIv2 session cache. Changing these values can avoid resource shortages. Refer to the Common Secure Interoperability Version 2 outbound communications topic for more information.
  - If you are communicating only with WebSphere Application Server Version 5 or higher servers, make the Active Authentication Protocol CSI, instead of CSI and SAS. This action removes an interceptor invocation for every request on both the client and server sides.

    Important: SAS is supported only between Version 6.0.x and previous version servers that have been federated in a Version 6.1 cell.
- Consider the following steps to tune Lightweight Directory Access Protocol (LDAP) authentication.
  1. In the administration console, click **Security > Global security**.
  2. Under User account repository, click the **Available realm definitions** drop-down list, select **Standalone LDAP registry** and click **Configure**.

3. Select the **Ignore case for authorization** option in the stand-alone LDAP registry configuration, when case-sensitivity is not important.
4. Select the **Reuse connection** option.
5. Use the cache features that your LDAP server supports.
6. Choose either the IBM Tivoli Directory Server or SecureWay directory type, if you are using an IBM Tivoli Directory Server. The IBM Tivoli Directory Server yields improved performance because it is programmed to use the new group membership attributes to improve group membership searches. However, authorization must be case insensitive to use IBM Tivoli Directory Server.
7. Choose either iPlanet Directory Server (also known as Sun ONE) or Netscape as the directory if you are an iPlanet Directory user. Using the iPlanet Directory Server directory can increase performance in group membership lookup. However, use **Role** only for group mechanisms.

- Consider the following steps to tune web authentication.
  - Increase the cache and token timeout values if you feel your environment is secure enough. The web authentication information is stored in these caches and as long as the authentication information is in the cache, the login module is not invoked to authenticate the user. This supports subsequent requests to reuse the credentials that are already created. A disadvantage of increasing the token timeout is the exposure of having a token stolen and providing the thief more time to hack into the system before the token expires.
  - Enable single sign-on (SSO). To configure SSO, click **Security > Global security**. Under Web security, click **Single sign-on (SSO)**.

    SSO is only available when you configure **LTPA** as the authentication mechanism in the Authentication mechanisms and expiration panel. Although you can select Simple WebSphere Authentication Mechanism (SWAM) as the authentication mechanism on the Authentication mechanisms and expiration panel, SWAM is deprecated in Version 8.5 and does not support SSO. When you select SSO, a single authentication to one application server is enough to make requests to multiple application servers in the same SSO domain. Some situations exist where SSO is not a desirable and you do not want to use it in those situations.
  - Disable or enable the **Web Inbound Security Attribute Propagation** option on the Single sign-on (SSO) panel if the function is not required. In some cases, having the function enabled can improve performance. This improvement is most likely for higher volume cases where a considerable number of user registry calls reduces performance. In other cases, having the feature disabled can improve performance. This improvement is most likely when the user registry calls do not take considerable resources.
  - The following two custom properties might help to improve performance when security attribute propagation is enabled:
    - **com.ibm.CSI.propagateFirstCallerOnly**

      The default value of this property is `true`. When this custom property is set to `true` the first caller in the propagation token that stays on the thread is logged when security attribute propagation is enabled. When this property is set to `false`, all of the caller switches are logged, which can affect performance.
    - **com.ibm.CSI.disablePropagationCallerList**

      When this custom property is set to `true` the ability to add a caller or host list in the propagation token is completely disabled. This function is beneficial when the caller or host list in the propagation token is not needed in the environment.
- Consider the following steps to tune authorization.
  - Map your users to groups in the user registry. Associate the groups with your Java Platform, Enterprise Edition (Java EE) roles. This association greatly improves performance when the number of users increases.
  - Judiciously assign method-permissions for enterprise beans. For example, you can use an asterisk (*) to indicate all the methods in the method-name element. When all the methods in enterprise beans require the same permission, use an asterisk (*) for the method-name to indicate all methods.

This indication reduces the size of deployment descriptors and reduces the memory that is required to load the deployment descriptor. It also reduces the search time during method-permission match for the enterprise beans method.

  - Judiciously assign security-constraints for servlets. For example, you can use the `*.jsp` URL pattern to apply the same authentication data constraints to indicate all JavaServer Pages (JSP) files. For a given URL, the exact match in the deployment descriptor takes precedence over the longest path match. Use the `*.jsp`, `*.do`, `*.html` extension match if no exact matches exist and longest path matches exist for a given URL in the security constraints.

- Use new tuning parameters when using Java 2 security. The new tuning parameters can improve performance significantly, and introduce a new concept called *Read-only Subject*, which enables a new cache for J2C Auth Subjects when using container-managed auth data aliases. If the J2C auth subject does not need to be modified after it is created, the following new tuning parameters can be used to improve Java 2 Security performance:
  - com.ibm.websphere.security.auth.j2c.cacheReadOnlyAuthDataSubjects=true
  - com.ibm.websphere.security.auth.j2c.readOnlyAuthDataSubjectCacheSize=50 (This is the maximum number of subjects in the hashtable of the cache. Once the cache reaches this size, some of the entries are purged. For better performance, this size should be equal to the number of unique subjects (cache based on uniqueness of user principal + auth data alias + managed connection factory instance) when role-based security and Java 2 security are used together).
- Use new tuning parameters to improve the performance of Security Attribute Propagation. The new tuning parameters can be set through custom properties in the administrative console to reduce the extra overhead of Security Attribute Propagation:
  - com.ibm.CSI.disablePropagationCallerList=true
  - com.ibm.CSI.propagateFirstCallerOnly=true (use if you want to track the first caller only).

Results

You always have a trade off between performance, feature, and security. Security typically adds more processing time to your requests, but for a good reason. Not all security features are required in your environment. When you decide to tune security, create a benchmark before making any change to ensure that the change is improving performance.

What to do next

In a large scale deployment, performance is very important. Running benchmark measurements with different combinations of features can help you to determine the best performance versus the benefit of configuration for your environment. Continue to run benchmarks if anything changes in your environment, to help determine the impact of these changes.

Secure Sockets Layer performance tips

Use this page to learn about Secure Sockets Layer (SSL) performance tips. Be sure to consider that performance issues typically involve trade-offs between function and speed. Usually, the more function and the more processing that are involved, the slower the performance.

The following are two types of Secure Sockets Layer (SSL) performance:
- Handshake
- Bulk encryption and decryption

When an SSL connection is established, an SSL handshake occurs. Once a connection is made, SSL performs bulk encryption and decryption for each read-write. The performance cost of an SSL handshake is much larger than that of bulk encryption and decryption.

To enhance SSL performance, decrease the number of individual SSL connections and handshakes.

Decreasing the number of connections increases performance for secure communication through SSL connections, as well as non-secure communication through simple Transmission Control Protocol/Internet Protocol (TCP/IP) connections. One way to decrease individual SSL connections is to use a browser that supports HTTP 1.1. Decreasing individual SSL connections can be impossible if you cannot upgrade to HTTP 1.1.

Another common approach is to decrease the number of connections (both TCP/IP and SSL) between two WebSphere Application Server components. The following guidelines help to verify the HTTP transport of the application server is configured so that the Web server plug-in does not repeatedly reopen new connections to the application server:

- Verify that the maximum number of keep alives are, at minimum, as large as the maximum number of requests per thread of the web server (or maximum number of processes for IBM HTTP Server on UNIX). Make sure that the web server plug-in is capable of obtaining a keep alive connection for every possible concurrent connection to the application server. Otherwise, the application server closes the connection once a single request is processed. Also, the maximum number of threads in the web container thread pool should be larger than the maximum number of keep alives, to prevent the keep alive connections from consuming the web container threads.

  Note: HTTP Transports have been deprecated. For instructions on how to set a maximum keep alive value for channel based configurations, see HTTP transport channel settings.
- Increase the maximum number of requests per keep alive connection. The default value is 100, which means the application server closes the connection from the plug-in following 100 requests. The plug-in then has to open a new connection. The purpose of this parameter is to prevent denial of service attacks when connecting to the application server and preventing continuous send requests to tie up threads in the application server.
- Use a hardware accelerator if the system performs several SSL handshakes.

  Hardware accelerators currently supported by WebSphere Application Server only increase the SSL handshake performance, not the bulk encryption and decryption. An accelerator typically only benefits the web server because Web server connections are short-lived. All other SSL connections in WebSphere Application Server are long-lived.
- Use an alternative cipher suite with better performance.

  The performance of a cipher suite is different with software and hardware. Just because a cipher suite performs better in software does not mean a cipher suite will perform better with hardware. Some algorithms are typically inefficient in hardware, for example, Data Encryption Standard (DES) and triple-strength DES (3DES); however, specialized hardware can provide efficient implementations of these same algorithms.

  The performance of bulk encryption and decryption is affected by the cipher suite used for an individual SSL connection. The following chart displays the performance of each cipher suite. The test software calculating the data was Java Secure Socket Extension (JSSE) for both the client and server software, which used no cryptographic hardware support. The test did not include the time to establish a connection, but only the time to transmit data through an established connection. Therefore, the data reveals the relative SSL performance of various cipher suites for long running connections.

  Prior to establishing a connection, the client enables a single cipher suite for each test case. After the connection is established, the client times how long it takes to write an integer to the server and for the server to write the specified number of bytes back to the client. Varying the amount of data had negligible effects on the relative performance of the cipher suites.

An analysis of the previous data reveals the following:

- Bulk encryption performance is only affected by what follows the WITH in the cipher suite name. This is expected since the portion preceding the WITH identifies the algorithm used only during the SSL handshake.
- MD5 and Secure Hash Algorithm (SHA) are the two hash algorithms used to provide data integrity. MD5 is generally faster than SHA, however, SHA is more secure than MD5.

- DES and RC2 are slower than RC4. Triple DES is the most secure, but the performance cost is high when using only software.
- The cipher suite providing the best performance while still providing privacy is SSL_RSA_WITH_RC4_128_MD5. Even though SSL_RSA_EXPORT_WITH_RC4_40_MD5 is cryptographically weaker than RSA_WITH_RC4_128_MD5, the performance for bulk encryption is the same. Therefore, as long as the SSL connection is a long-running connection, the difference in the performance of high and medium security levels is negligible. It is recommended that a security level of high be used, instead of medium, for all components participating in communication only among WebSphere Application Server products. Make sure that the connections are long running connections.

Tuning security performance

Use the following procedures to tune the performance, without compromising your security settings.

About this task

Enabling security decreases performance. The following tuning parameters provide ways to minimize this performance impact.

Procedure

- Disable security on any application servers that do not need security. You can disable security in the administrative console by clicking **Security > Global security** and deselecting the **Enable administrative security** option.
- Fine-tune the **Authentication cache timeout** value on the Authentication mechanisms and expiration panel in the administrative console. For more information, see the “Global security settings” on page 86 topic.
- Configure the security cache properties. For more information, see the “Authentication cache settings” on page 157 topic.
- Enable the **Enable SSL ID tracking** option on the Session management panel in the administrative console.
- Improve the performance of Web Services Security by downloading a Java Cryptography Extension (JCE) unlimited jurisdiction policy file that does not have restrictions on cryptography strength. See the information about tuning Web Services Security for Version 8.0 applications for details.
- Read the Secure Sockets Layer performance tips and “Tuning security configurations” on page 975 topics for more information.

Hardening security configurations

There are several methods that you can use to protect the WebSphere Application Server infrastructure and applications from different forms of attack. Several different techniques can help with multiple forms of attack. Sometimes a single attack can leverage multiple forms of intrusion to achieve the end goal.

About this task

For example, in the simplest case, network sniffing can be used to obtain passwords and those passwords can then be used to mount an application-level attack. The following issues are discussed in IBM WebSphere Developer Technical Journal: WebSphere Application Server V5 advanced security and system hardening:

Procedure

- Take preventative measures to protect the infrastructure.
- Make applications less vulnerable to attack.

- At a minimum, ensure administrative security is enabled in all WebSphere processes. This protects access to the administrative ConfigService interface and managed beans (MBeans) that enables control over the WebSphere process if it is compromised.
- Ensure Secure Sockets Layer (SSL) is used whenever possible, and mutual SSL whenever possible. However, mutual SSL requires all clients to supply a trusted personal certificate in order to connect.
- Remove any unnecessary certificate authority (CA) signer certificates from your trust stores.
- Change default keystore passwords during or after profile creation using the `AdminTask changeMultipleKeyStorePasswords` command.
- Change your Lightweight Third-Party Authentication (LTPA) keys periodically. You can configure the automatic regeneration of LTPA keys if necessary.
- Common Secure Interoperability version 2 (CSIv2) inbound Basic authentication is supported in this release of WebSphere Application Server. The authentication default is 'required'.

What to do next

Note: In this release of WebSphere Application Server, more security hardening features of the server are enabled by default. However, if the features are not enabled after migration you can enable them yourself. See the Security hardening features enablement and migration article for more information.

For additional information about hardening security configurations, see the WebSphere Application Server security web page.

Enablement and migration considerations of Security hardening features

In this release of WebSphere Application Server, more security hardening features of the server are enabled out-of-the-box by default. When migrating, the settings that were enabled prior to migration are retained. However, if the features are not enabled after migration you can enable them yourself.

To ensure that WebSphere Application Server configuration is set to be secure by default, the following defaults have been changed as part of the new security hardening features in WebSphere Application Server Version 8.0:

- Enablement of Secure Sockets Layer (SSL)-required on Common Secure Interoperability version 2 (CSIv2) transport by default

  The following settings for the CSIv2 transport layer exist: TCP/IP for a TCP/IP connection, SSL-supported for a TCP/IP or an SSL connection, and SSL-required for an SSL connection only. SSL-required is the new default in this release of WebSphere Application Server. Switching to SSL-required as the default setting ensures that all CSIv2 connections into and out of the server are using the secure SSL connection.
- Enablement of the HttpOnly attribute on LTPA cookies by default

  When the com.ibm.ws.security.addHttpOnlyAttributeToCookies custom property is set to `true`, the HttpOnly attribute is added to those security cookies (LTPA and WASReqURL cookies) that are created by the server. The HttpOnly attribute is a browser attribute created to prevent client side applications (such as Java scripts) from accessing cookies to prevent some cross-site scripting vulnerabilities. This attribute is now configurable in the administrative console. Prior to WebSphere Application Server Version 8.0, the com.ibm.ws.security.addHttpOnlyAttributeToCookies custom property default was `false`. For WebSphere Application Server Version 8.0, the default is now `true` for both the LTPA cookie and the Session Cookie.

  For more information see the custom property com.ibm.ws.security.addHttpOnlyAttributeToCookies in the Security custom properties article.
- Enablement of session security integration by default

Only authenticated users can access sessions created in secure pages. The session management facility uses the security infrastructure to determine the authenticated identity associated with a client HTTP request, and either retrieves or creates a session. For more information on session security, read the Session security support article.

Along with enabling session security integration, credential persistence is enabled as well. This allows login information to be available to unprotected web clients to enable additional access to user information. For more information on credential persistence, see the “Use available authentication data when an unprotected URI is accessed” feature in the web authentication settings article.

Enabling the new security hardening features after migration

If the new security features are not enabled after migration, you can enable them yourself using the administrative console or by scripting.

Enablement of SSL by default on CSIv2

To enable SSL by default for inbound and outbound transports on CSIv2:

If you are using the administrative console, select **Security** > **Global security** > **RMI/IIOP** > **CSIv2 inbound communications**. In the Transport box, select **SSL- required** from the pull-down list and then click **Apply**.

Repeat the same steps for CSIv2 outbound communications and click **Security** > **Global security** > **RMI/IIOP** > **CSIv2 outbound communications**. In the Transport box, select **SSL- required** from the menu list and then click **Apply**.

If you want to enable SSL by default for inbound and outbound transports on CSIv2 using scripting, use the configureCSIInbound and configureCSIOutbound commands. See the Configuring Common Secure Interoperability authentication using scripting topic for more information.

For the client side, edit the sas.client.props file. Change com.ibm.CSI.performTransportAssocSSLTLSRequired to `true` and change com.ibm.CSI.performTransportAssocSSLTLSSupported to `false`.

Enablement of the HttpOnly cookie attribute

To enable the HttpOnly attribute on cookies attribute by default:

If you are using the administrative console, click **Security** > **Global security** > **Custom properties**. Click**New** and enter `com.ibm.ws.security.addHttpOnlyAttributeToCookies` for the Name and `true` for the Value.

You can also enable the **HttpOnly** attribute using the administrative console by clicking **Security** > **Global security** > **Single sign-on (SSO)**. Click Set security cookies to **HTTPOnly** to help prevent cross-site scripting attacks, and then click **Apply**.

To enable the HttpOnly attribute on cookies attribute by default using scripting, use the setAdminActiveSecuritySettings command.

Enablement of session security integration

To enable session security integration for each server by using the administrative console, select **Servers** > **Server types** > **WebSphere application servers** > **server1** > **Session management**. Select the **security integration** check box.

To enable persisting credentials from the administrative console, click **Security** > **Global security** > **Web and SIP security** > **General settings**. Select the **Use available authentication data when an unprotected URI is accessed** check box.

Security hardening features enablement troubleshooting

When the new security hardening features are enabled you might see some differences in system behavior depending upon which environment you might have used in the past.

For example, if you are coming from an environment where CSIv2 transport was set to the previous default of SSL-supported, you do not experience any differences, as SSL-supported communicates with both TCP/IP and SSL connections. If a problem is encountered, however, certificates might not have been exchanged correctly to enable the client and server to communicate. Read about the Secure communications using Secure Sockets Layer (SSL) topic for more information.

If you worked in an environment where TCP/IP is used for the connection to CSIv2, you might experience connection problem to the SSL-enabled CSIv2 connection. The server configuration can be modified to SSL-supported or to TCP/IP if SSL is not required.

For the HttpOnly attribute, when the attribute is added to the security cookies, the browser prevents client side scripts from accessing these cookies. In most case this should be the default behavior to minimize cross-site scripting vulnerabilities. If there is an absolute need to allow client-side scripts to access WebSphere security cookies, and you are aware of the possible consequences, then the setting of the HttpOnly attribute can be disabled.

However, the HttpOnly attribute can possibly uncover client-side scripts that are used to access WebSphere cookies, and can then use them even though it was not intended to do so. If this happens, the web application that enables the scripts to access the WebSphere cookies must be evaluated.

For session security integration enablement, when session integrated security is enabled you might receive an UnauthorizedSessionRequestException exception on servlets if they access a session that belongs to authenticated identities other than to the identity that currently owns the session. If you do not want this checking to occur, you can disable session security from the server that is experiencing the problem.

Securing passwords in files

Password encoding and encryption deters the casual observation of passwords in server configuration and property files.

About this task

The following topics can be used to add protection for passwords located in files:

Procedure

- Encoding passwords in files WebSphere Application Server contains some encoded passwords that are not encrypted. The **PropFilePasswordEncoder** utility is included to encode these passwords. For more information on encoding passwords in a file, see “Encoding passwords in files.”
- Enabling custom password encryption You need to protect passwords that are contained in your WebSphere Application Server configuration. You can added protection by creating a custom class for encrypting the passwords. For more information on custom password encryption, see “Enabling custom password encryption” on page 987.

Encoding passwords in files

The purpose of password encoding is to deter casual observation of passwords in server configuration and property files. Use the **PropFilePasswordEncoder** utility to encode passwords stored in properties files. WebSphere Application Server does not provide a utility for decoding the passwords. Encoding is not sufficient to fully protect passwords. Native security is the primary mechanism for protecting passwords used in WebSphere Application Server configuration and property files.

About this task

WebSphere Application Server contains several encoded passwords in files that are not encrypted. WebSphere Application Server provides the **PropFilePasswordEncoder** utility, which you can use to

encode passwords. The purpose of password encoding is to deter casual observation of passwords in server configuration and property files. The **PropFilePasswordEncoder** utility does not encode passwords that are contained within XML or XMI files.

Table 164. XML and XMI files that contain encoded passwords. Instead, WebSphere Application Server automatically encodes the passwords in these files. XML and XMI files that contain encoded passwords include the following:

File name	Additional information
profile_root/config/cells/*cell_name*/security.xml	The following fields contain encoded passwords: • LTPA password • JAAS authentication data • User registry server password • LDAP user registry bind password • Keystore password • Truststore password • Cryptographic token device password
war/WEB-INF/ibm_web_bnd.xml	Specifies the passwords for the default basic authentication for the resource-ref bindings within all the descriptors, except in the Java cryptography architecture
ejb jar/META-INF/ibm_ejbjar_bnd.xml	Specifies the passwords for the default basic authentication for the resource-ref bindings within all the descriptors, except in the Java cryptography architecture
client jar/META-INF/ibm-appclient_bnd.xml	Specifies the passwords for the default basic authentication for the resource-ref bindings within all the descriptors, except in the Java cryptography architecture
ear/META-INF/ibm_application_bnd.xml	Specifies the passwords for the default basic authentication for the run as bindings within all the descriptors
profile_root/config/cells/*cell_name* /nodes/*node_name*/servers/ *server_name*/security.xml	The following fields contain encoded passwords: • Keystore password • Truststore password • Cryptographic token device password • Session persistence password
profile_root/config/cells/*cell_name* /nodes/*node_name*/servers/ *server_name*/resources.xml	The following fields contain encoded passwords: • WAS40Datasource password • mailTransport password • mailStore password • MQQueue queue mgr password
• *profile_root*/config/cells/*cell_name* /ws-security.xml • *profile_root*/config/cells/*cell_name* /nodes/*node_name*/servers/*server_name*/ws-security	
ibm-webservices-bnd.xmi	
ibm-webservicesclient-bnd.xmi	

Table 165. The PropFilePasswordEncoder utility - Partial File List. You use the **PropFilePasswordEncoder** *utility to encode the passwords in properties files. These files include:*

File name	Additional information
profile_root /properties/sas.client.props	Specifies the passwords for the following files: • com.ibm.ssl.keyStorePassword • com.ibm.ssl.trustStorePassword • com.ibm.CORBA.loginPassword
profile_root /properties/sas.tools.properties	Specifies passwords for: • com.ibm.ssl.keyStorePassword • com.ibm.ssl.trustStorePassword • com.ibm.CORBA.loginPassword
profile_root /properties/sas.stdclient.properties	Specifies passwords for: • com.ibm.ssl.keyStorePassword • com.ibm.ssl.trustStorePassword • com.ibm.CORBA.loginPassword
profile_root /properties/wsserver.key	
profile_root/profiles/AppSrvXX/properties/sib.client.ssl.properties	Specifies passwords for: • com.ibm.ssl.keyStorePassword • com.ibm.ssl.trustStorePassword

Table 165. The PropFilePasswordEncoder utility - Partial File List (continued). You use the **PropFilePasswordEncoder** *utility to encode the passwords in properties files. These files include:*

File name	Additional information
profile_root/UDDIReg/scripts/UDDIUtilityTools.properties	Specifies passwords for: • trustStore.password

To encode a password again in one of the previous files, complete the following steps:

Procedure

1. Access the file using a text editor and type over the encoded password. The new password is shown is no longer encoded and must be re-encoded.
2. Use the `PropFilePasswordEncoder.bat` or the `PropFilePasswordEncode.sh` file in the *profile_root*`/bin` directory to encode the password again.

   If you are encoding files that are not SAS properties files, type `PropFilePasswordEncoder` *"file_name" password_properties_list*

   Important: When you use the **PropFilePasswordEncoder** utility, a prompt asks whether a backup version of the original file is required. If a backup version is required, a backup file (.bak), is created with the clear text password. Examine the results and then delete this backup file. It contains the unencrypted password. If you do not want to see this prompt, edit the PropFilePasswordEncoder utility and add the following Java system property as a parameter: `-Dcom.ibm.websphere.security.util.createBackup=true` or `-Dcom.ibm.websphere.security.util.createBackup=false`

   A `true` value for the Java system property creates a backup file and a `false` value disables the backup file.

   where:

   "file_name" is the name of the z/SAS properties file, and *password_properties_list* is the name of the properties to encode within the file.

   Note: Only the password should be encoded in this file using the **PropFilePasswordEncoder** tool.

   Use the **PropFilePasswordEncoder** utility to encode WebSphere Application Server password files only. The utility cannot encode passwords that are contained in XML files or other files that contain open and close tags. To change passwords in these files, use the administrative console or an assembly tool such as the Rational Application Developer.

Results

If you reopen the affected files, the passwords are encoded. WebSphere Application Server does not provide a utility for decoding the passwords.

Example

The following example shows how to use the **PropFilePasswordEncoder** tool:

```
PropFilePasswordEncoder C:\WASV8\WebSphere\AppServer\profiles\AppSrv\properties
\sas.client.props com.ibm.ssl.keyStorePassword,com.ibm.ssl.trustStorePassword
```

where:

PropFilePasswordEncoder is the name of the utility that you are running from the profile_root/profiles/ profile_name/bin directory.

`C:\WASV6\WebSphere\AppServer\profiles\AppSrv\properties\sas.client.props` is the name of the file that contains the passwords to encode.

com.ibm.ssl.keyStorePassword is a password to encode in the file.

com.ibm.ssl.trustStorePassword is a second password to encode in the file.

PropFilePasswordEncoder command reference

The **PropFilePasswordEncoder** command encodes passwords that are located in plain text property files. This command encodes both Secure Authentication Server (SAS) property files and non-SAS property files. After you encode the passwords, a decoding command does not exist.

Note: If you need to custom encode passwords in property files, manually edit the PropFilePasswordEncoder.sh or PropFilePasswordEncoder.bat file before issuing this command. See the topic Implementing custom password encryption for a description of the lines that need to be added to this file.

Note: To enable PropFilePasswordEncoder to print out more a debug message than in previous releases, update the command by entering the following:

```
-Dcom.ibm.websphere.security.passwordEncoderDebug=true
```

Syntax

The command syntax is as follows:

```
PropFilePasswordEncoder "file_name" { passwordPropertiesList | -SAS } { -noBackup | -Backup }
    [ -profileName profile ] [ -help | -? ]
```

Parameters

The following option is available for the **PropFilePasswordEncoder** command:

file_name
This required parameter specifies the name of the file in which passwords are encoded.

passwordPropertiesList
This parameter is required if you are encoding passwords in property files other than the `sas.client.props` file. Specify one or more password properties that you want to encode. The password properties list should be delimited by commas.

-SAS
This parameter is required if you are encoding passwords in the `sas.client.props` file.

-noBackup
This parameter is optional and the default. The script does not create a backup file. The default value can be altered by adding following Java System Property: `"-Dcom.ibm.websphere.security.util.createBackup=true"`.

-Backup
This parameter is optional. The script creates a backup file, `<file_name>.bak`, which contains passwords in clear text.

-profileName
This parameter is optional. The profile value specifies an application server profile name. The script uses the password encoding algorithm that it retrieves from the specified profile. If you do not specify this parameter, the script uses the default profile.

-help or -?
If you specify this parameter, the script ignores all other parameters and displays usage text.

Enabling custom password encryption

You need to protect passwords that are contained in your WebSphere Application Server configuration. After creating your server profile, you can added protection by creating a custom class for encrypting the passwords.

Before you begin

Create your custom class for encrypting passwords. For more information, see “Plug point for custom password encryption” on page 926.

About this task

Complete the following steps to enable custom password encryption.

Procedure

1. Add the following system properties for every server and client process. For server processes, update the `server.xml` file for each process. Add these properties as a genericJvmArgument argument preceded by a **-D** prefix.

   ```
   com.ibm.wsspi.security.crypto.customPasswordEncryptionClass=
          com.acme.myPasswordEncryptionClass
   com.ibm.wsspi.security.crypto.customPasswordEncryptionEnabled=true
   ```

   Tip: If the custom encryption class name is com.ibm.wsspi.security.crypto.CustomPasswordEncryptionImpl, it is automatically enabled when this class is present in the classpath. Do not define the system properties that are listed previously when the custom implementation has this package and class name. To disable encryption for this class, you must specify `com.ibm.wsspi.security.crypto.customPasswordEncryptionEnabled=false` as a system property.
2. Choose one of the following methods to configure the WebSphere Application Server runtime to load the custom encryption implementation class:
   - Place the custom encryption class in a Java archive (JAR) file that resides in the `${WAS_INSTALL_ROOT}/classes` directory, which you have created.

     gotcha: WebSphere Application Server does not create the `${WAS_INSTALL_ROOT}/classes` directory. For more information on the classes directory, see the topic, "Creating a classes subdirectory in your profile for custom classes".
   - Place the custom encryption class in a Java archive (JAR) file that resides in the `${WAS_HOME}/lib/ext` directory.
3. Restart all server processes.
4. Edit each configuration document that contains a password and save the configuration. All password fields are then run through the **WSEncoderDecoder** utility, which calls the plug point when it is enabled. The `{custom:alias}` tags are displayed in the configuration documents. The passwords, even though they are encrypted, are still Base64-encoded. They seem similar to encoded passwords, except for the tags difference.
5. Encrypt any passwords that are in client-side property files using the **PropsFilePasswordEncoder** (.bat or .sh) utility. This utility requires that the properties listed previously are defined as system properties in the script to encrypt new passwords instead of encoding them.
6. To decrypt passwords from client Java virtual machines (JVMs), add the properties listed previously as system properties for each client utility.
7. Ensure that all nodes have the custom encryption classes in their class paths prior to enabling this function.

Results

Custom password encryption is enabled.

What to do next

If custom password encryption fails or is no longer required, see “Disabling custom password encryption.”

Disabling custom password encryption

If custom password encryption fails or is no longer required, perform this task to disable custom password encryption.

Before you begin

Enable custom password encryption.

About this task

Complete the following steps to disable custom password encryption.

Procedure

1. Change the com.ibm.wsspi.security.crypto.customPasswordEncryptionEnabled property to be `false` in the `security.xml` file, but leave the com.ibm.wsspi.security.crypto.customPasswordEncryptionClass property configured. Any passwords in the model that still have the {`custom:alias`} tag are decrypted by using the customer password encryption class.
2. If an encryption key is lost, any passwords that are encrypted with that key cannot be retrieved. To recover a password, retype the password in the password field in plaintext and save the document. The new password must be written out using encoding with the {xor} tag with scripting or from the administrative console.

   ```
   com.ibm.wsspi.security.crypto.customPasswordEncryptionClass=
          com.acme.myPasswordEncryptionClass
   com.ibm.wsspi.security.crypto.customPasswordEncryptionEnabled=false
   ```
3. Restart all processes to make the changes effective.
4. Edit each configuration document that contains an encrypted password and save the configuration. All password fields are then run through the **WSEncoderDecoder** utility, which calls the plug point in the presence of the {`custom:alias`} tag. The {`xor`} tags display in the configuration documents again after the documents are saved.
5. Decrypt and encode any passwords that are in client-side property files using the **PropsFilePasswordEncoder** (.bat or .sh) utility. If the encryption class is specified, but custom encryption is disabled, running this utility converts the encryption to encoding and causes the {`xor`} tags to display again.
6. Disable custom password encryption from the client Java virtual machines (JVMs) by adding the system properties listed previously to all client scripts. This action enables the code to decrypt passwords, but this action is not used to encrypt them again. The {`xor`} algorithm becomes the default for encoding. Leave the custom password encryption class defined for a time in case any encrypted passwords still exist in the configuration.

Results

Custom password encryption is disabled.

Chapter 13. Troubleshooting security configurations

The following topics help to troubleshoot specific problems that are related to configuring and enabling security configurations.

About this task

Refer to Security components troubleshooting tips for instructions on how to troubleshoot errors that are related to security.

Refer to SPNEGO TAI troubleshooting tips for instructions on how to troubleshoot errors that are related to diagnosing Simple and Protected GSS-API Negotiation (SPNEGO) trust association interceptor (TAI) problems and exceptions.

Note:

In WebSphere Application Server Version 6.1, a trust association interceptor (TAI) that uses the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) to securely negotiate and authenticate HTTP requests for secured resources was introduced. This function was deprecated in WebSphere Application Server 7.0. SPNEGO web authentication has taken its place to provide dynamic reload of the SPNEGO filters and to enable fallback to the application login method.

Procedure

- Errors when configuring or enabling security
- Errors after enabling security
- Access problems after enabling security
- Errors after configuring or enabling Secure Sockets Layer
- Errors configuring Secure Sockets Layer encrypted access
- Single sign-on configuration troubleshooting tips
- Authorization provider troubleshooting tips

Security components troubleshooting tips

This document explains basic resources and steps for diagnosing security-related issues in WebSphere Application Server.

Note: This topic references one or more of the application server log files. As a recommended alternative, you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log` , `SystemErr.log`, `trace.log`, and `activity.log` files on distributed and IBM i systems. You can also use HPEL in conjunction with your native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

Basic resources and steps for diagnosing security-related issues in WebSphere Application Server include:
- What “Log files” on page 990 to look at and what to look for in them.
- What to look at and what to look for “Using SDSF” on page 991.
- “General approach for troubleshooting security-related issues” on page 992 to isolating and resolving security problems.
- When and how to “Trace security” on page 995.
- An overview and table of “CSIv2 CORBA minor codes” on page 997.

The following security-related problems are addressed elsewhere in the information center:

- Errors and access problems after enabling security

  After enabling security, a degradation in performance is realized. For more information about using unrestricted policy files, see the Enabling security for the realm section of the *Securing applications and their environment* PDF book.
- Errors after enabling SSL, or SSL-related error messages
- Errors trying to configure and enable security

If none of these steps solves the problem, check to see if the problem is identified and documented using the links in the Diagnosing and fixing problems: Resources for learning article.

If you do not see a problem that resembles yours, or if the information provided does not solve your problem, see Troubleshooting help from IBM for further assistance.

For an overview of WebSphere Application Server security components such as Secure Authentication Services (SAS) and how they work in a distributed or an iSeries environment, refer to the *Securing applications and their environment* PDF book.

Important: SAS is supported only between Version 6.0.x and previous version servers that have been federated in a Version 6.1 cell.

Log files

When troubleshooting the security component, browse the Java Virtual Machine (JVM) logs for the server that hosts the resource you are trying to access. The following is a sample of messages you would expect to see from a server in which the security service has started successfully:

```
SASRas        A CWWSA0001I: Security configuration initialized.
SASRas        A CWWSA0002I: Authentication protocol: CSIV2/IBM
SASRas        A CWWSA0003I: Authentication mechanism: SWAM
SASRas        A CWWSA0004I: Principal name: MYHOSTNAME/aServerID
SASRas        A CWWSA0005I: SecurityCurrent registered.
SASRas        A CWWSA0006I: Security connection interceptor initialized.
SASRas        A CWWSA0007I: Client request interceptor registered.
SASRas        A CWWSA0008I: Server request interceptor registered.
SASRas        A CWWSA0009I: IOR interceptor registered.
NameServerImp I CWNMS0720I: Do Security service listener registration.
SecurityCompo A CWSCJ0242A: Security service is starting
UserRegistryI A CWSCJ0136I: Custom Registry:com.ibm.ws.security.registry.nt.
NTLocalDomainRegistryImpl has been initialized
SecurityCompo A CWSCJ0202A: Admin application initialized successfully
SecurityCompo A CWSCJ0203A: Naming application initialized successfully
SecurityCompo A CWSCJ0204A: Rolebased authorizer initialized successfully
SecurityCompo A CWSCJ0205A: Security Admin mBean registered successfully
SecurityCompo A CWSCJ0243A: Security service started successfully
SecurityCompo A CWSCJ0210A: Security enabled true
```

The following is an example of messages from a server which cannot start the security service, in this case because the administrative user ID and password given to communicate with the user registry is wrong, or the user registry itself is down or misconfigured:

```
SASRas        A CWWSA0001I: Security configuration initialized.
SASRas        A CWWSA0002I: Authentication protocol: CSIV2/IBM
SASRas        A CWWSA0003I: Authentication mechanism: SWAM
SASRas        A CWWSA0004I: Principal name: MYHOSTNAME/aServerID
SASRas        A CWWSA0005I: SecurityCurrent registered.
SASRas        A CWWSA0006I: Security connection interceptor initialized.
SASRas        A CWWSA0007I: Client request interceptor registered.
SASRas        A CWWSA0008I: Server request interceptor registered.
SASRas        A CWWSA0009I: IOR interceptor registered.
NameServerImp I CWNMS0720I: Do Security service listener registration.

SecurityCompo A CWSCJ0242A: Security service is starting
UserRegistryI A CWSCJ0136I: Custom Registry:com.ibm.ws.security.
registry.nt.NTLocalDomainRegistryImpl has been initialized
Authenticatio E CWSCJ4001E: Login failed for badID/<null>
javax.security.auth.login.LoginException: authentication failed: bad user/password
```

The following is an example of messages from a server for which Lightweight Directory Access Protocol (LDAP) has been specified as the security mechanism, but the LDAP keys have not been properly configured:

```
SASRas        A CWWSA0001I: Security configuration initialized.
SASRas        A CWWSA0002I: Authentication protocol: CSIV2/IBM
SASRas        A CWWSA0003I: Authentication mechanism: LTPA
SASRas        A CWWSA0004I: Principal name: MYHOSTNAME/anID
SASRas        A CWWSA0005I: SecurityCurrent registered.
SASRas        A CWWSA0006I: Security connection interceptor initialized.
SASRas        A CWWSA0007I: Client request interceptor registered.
SASRas        A CWWSA0008I: Server request interceptor registered.
SASRas        A CWWSA0009I: IOR interceptor registered.
NameServerImp I CWNMS0720I: Do Security service listener registration.
SecurityCompo A CWSCJ0242A: Security service is starting
UserRegistryI A CWSCJ0136I: Custom Registry:com.ibm.ws.security.registry.nt.
NTLocalDomainRegistryImpl has been initialized
SecurityServe E CWSCJ0237E: One or more vital LTPAServerObject configuration
attributes are null or not available. The attributes and values are password :
LTPA password does exist, expiration time 30, private key <null>, public key <null>,
and shared key <null>.
```

A problem with the Secure Sockets Layer (SSL) configuration might lead to the following message. Ensure that the keystore location and keystore passwords are valid. Also, ensure the keystore has a valid personal certificate and that the personal certificate public key or certificate authority (CA) root has been extracted on put into the truststore.

```
SASRas        A CWWSA0001I: Security configuration initialized.
SASRas        A CWWSA0002I: Authentication protocol: CSIV2/IBM
SASRas        A CWWSA0003I: Authentication mechanism: SWAM
SASRas        A CWWSA0004I: Principal name: MYHOSTNAME/aServerId
SASRas        A CWWSA0005I: SecurityCurrent registered.
SASRas        A CWWSA0006I: Security connection interceptor initialized.
SASRas        A CWWSA0007I: Client request interceptor registered.
SASRas        A CWWSA0008I: Server request interceptor registered.
SASRas        A CWWSA0009I: IOR interceptor registered.
SASRas        E CWWSA0026E: [SecurityTaggedComponentAssistorImpl.register]
Exception connecting object to the ORB.  Check the SSL configuration to ensure
 that the SSL keyStore and trustStore properties are set properly.  If the problem
persists, contact support for assistance. org.omg.CORBA.OBJ_ADAPTER:
ORB_CONNECT_ERROR (5) - couldn't get Server Subcontract  minor code:
4942FB8F  completed: No
```

Using SDSF

When troubleshooting the security component, use System Display and Search Facility (SDSF) to browse logs for the server that hosts the resource you are trying to access. The following sample of messages helps you see from a server in which the security service has started successfully:

```
+BBOM0001I com_ibm_authMechanisms_type_OID: No OID for this mechanism.
+BBOM0001I com_ibm_security_SAF_unauthenticated: WSGUEST.
+BBOM0001I com_ibm_security_SAF_EJBROLE_Audit_Messages_Suppress: 0.
+BBOM0001I com_ibm_ws_logging_zos_errorlog_format_cbe: NOT SET, 280
DEFAULT=0.
+BBOM0001I com_ibm_CSI_performClientAuthenticationRequired: 0.
+BBOM0001I com_ibm_CSI_performClientAuthenticationSupported: 1.
+BBOM0001I com_ibm_CSI_performTransportAssocSSLTLSRequired: 0.
+BBOM0001I com_ibm_CSI_performTransportAssocSSLTLSSupported: 1.
+BBOM0001I com_ibm_CSI_rmiInboundPropagationEnabled: 1.
+BBOM0001I com_ibm_CSI_rmiOutboundLoginEnabled: 0.
+BBOM0001I com_ibm_CSI_rmiOutboundPropagationEnabled: 1.
+BBOM0001I security_assertedID_IBM_accepted: 0.
+BBOM0001I security_assertedID_IBM_sent: 0.
+BBOM0001I security_disable_daemon_ssl: NOT SET, DEFAULT=0.
+BBOM0001I security_sslClientCerts_allowed: 0.
+BBOM0001I security_sslKeyring: NOT SET.
+BBOM0001I security_zOS_domainName: NOT SET.
+BBOM0001I security_zOS_domainType: 0.
+BBOM0001I security_zSAS_ssl_repertoire: SY1/DefaultIIOPSSL.
```

```
+BBOM0001I security_EnableRunAsIdentity: 0.
+BBOM0001I security_EnableSyncToOSThread: 0.
+BBOM0001I server_configured_system_name: SY1.
+BBOM0001I server_generic_short_name: BBOC001.
+BBOM0001I server_generic_uuid:  457
*** Message beginning with BBOO0222I apply to Java within ***
*** WebSphere Application Server Security ***
+BBOO0222I: SECJ6004I: Security Auditing is disabled.
+BBOO0222I: SECJ0215I: Successfully set JAAS login provider 631
configuration class to com.ibm.ws.security.auth.login.Configuration.
+BBOO0222I: SECJ0136I: Custom 632
Registry:com.ibm.ws.security.registry.zOS.SAFRegistryImpl has been initialized
+BBOO0222I: SECJ0157I: Loaded Vendor AuthorizationTable: 633
com.ibm.ws.security.core.SAFAuthorizationTableImpl
```

General approach for troubleshooting security-related issues

When troubleshooting security-related problems, the following questions are very helpful:

Does the problem occur when security is disabled?

This question is a good litmus test to determine that a problem is security related. However, just because a problem only occurs when security is enabled does not always make it a security problem. More troubleshooting is necessary to ensure the problem is really security-related.

Did security seem to initialize properly?

A lot of security code is visited during initialization. So you can see problems there first if the problem is configuration related.

The following sequence of messages that are generated in the `SystemOut.log` indicate normal code initialization of an application server. This sequence varies based on the configuration, but the messages are similar:

```
SASRas        A CWWSA0001I: Security configuration initialized.
SASRas        A CWWSA0002I: Authentication protocol: CSIV2/IBM
SASRas        A CWWSA0003I: Authentication mechanism: SWAM
SASRas        A CWWSA0004I: Principal name: BIRKT20/pbirk
SASRas        A CWWSA0005I: SecurityCurrent registered.
SASRas        A CWWSA0006I: Security connection interceptor initialized.
SASRas        A CWWSA0007I: Client request interceptor registered.
SASRas        A CWWSA0008I: Server request interceptor registered.
SASRas        A CWWSA0009I: IOR interceptor registered.
NameServerImp I CWNMS0720I: Do Security service listener registration.
SecurityCompo A CWSCJ0242A: Security service is starting
UserRegistryI A CWSCJ0136I: Custom Registry:com.ibm.ws.security.registry.nt.
NTLocalDomainRegistryImpl has been initialized
SecurityCompo A CWSCJ0202A: Admin application initialized successfully
SecurityCompo A CWSCJ0203A: Naming application initialized successfully
SecurityCompo A CWSCJ0204A: Rolebased authorizer initialized successfully
SecurityCompo A CWSCJ0205A: Security Admin mBean registered successfully
SecurityCompo A CWSCJ0243A: Security service started successfully

SecurityCompo A CWSCJ0210A: Security enabled true
```

The following sequence of messages generated in the SDSF active log indicate normal code initialization of an application server. Non-security messages have been removed from the sequence that follows. This sequence will vary based on the configuration, but the messages are similar:

```
Trace: 2005/05/06 17:27:31.539 01 t=8E96E0 c=UNK key=P8 (13007002)
  ThreadId: 0000000a
  FunctionName: printProperties
  SourceId: com.ibm.ws390.orb.CommonBridge
  Category: AUDIT
  ExtendedMessage: BBOJ0077I java.security.policy =
         /WebSphere/V6R1M0/AppServer/profiles/default/pr
Trace: 2005/05/06 17:27:31.779 01 t=8E96E0 c=UNK key=P8 (13007002)
  ThreadId: 0000000a
  FunctionName: printProperties
  SourceId: com.ibm.ws390.orb.CommonBridge
  Category: AUDIT
  ExtendedMessage: BBOJ0077I java.security.auth.login.config =
         /WebSphere/V6R1M0/AppServer/profiles/default/pr
Trace: 2005/05/06 17:27:40.892 01 t=8E96E0 c=UNK key=P8 (13007002)
  ThreadId: 0000000a
  FunctionName: com.ibm.ws.security.core.SecurityDM
```

```
  SourceId: com.ibm.ws.security.core.SecurityDM
  Category: INFO
  ExtendedMessage: BBOO0222I: SECJ0231I: The Security component's FFDC
         Diagnostic Module com.ibm.ws.security.core.Secur
red successfully: true.
Trace: 2005/05/06 17:27:40.892 01 t=8E96E0 c=UNK key=P8 (0000000A)
  Description: Log Boss/390 Error
  from filename: ./bborjtr.cpp
  at line: 932
  error message: BBOO0222I: SECJ0231I: The Security component's FFDC
         Diagnostic Module com.ibm.ws.security.core.Securit
d successfully: true.
Trace: 2005/05/06 17:27:41.054 01 t=8E96E0 c=UNK key=P8 (13007002)
  ThreadId: 0000000a
  FunctionName: com.ibm.ws.security.audit.AuditServiceImpl
  SourceId: com.ibm.ws.security.audit.AuditServiceImpl
  Category: AUDIT
  ExtendedMessage: BBOO0222I: SECJ6004I: Security Auditing is disabled.
Trace: 2005/05/06 17:27:41.282 01 t=8E96E0 c=UNK key=P8 (13007002)
  ThreadId: 0000000a
  FunctionName: com.ibm.ws.security.core.distSecurityComponentImpl
  SourceId: com.ibm.ws.security.core.distSecurityComponentImpl
  Category: INFO
  ExtendedMessage: BBOO0222I: SECJ0309I: Java 2 Security is disabled.
Trace: 2005/05/06 17:27:41.282 01 t=8E96E0 c=UNK key=P8 (0000000A)
  Description: Log Boss/390 Error
  from filename: ./bborjtr.cpp
  at line: 932
  error message: BBOO0222I: SECJ0309I: Java 2 Security is disabled.
Trace: 2005/05/06 17:27:42.239 01 t=8E96E0 c=UNK key=P8 (13007002)
  ThreadId: 0000000a
  FunctionName: com.ibm.ws.security.auth.login.Configuration
  SourceId: com.ibm.ws.security.auth.login.Configuration
  Category: AUDIT
  ExtendedMessage: BBOO0222I: SECJ0215I: Successfully set JAAS login
         provider configuration class to com.ibm.ws.securit
Configuration.
Trace: 2005/05/06 17:27:42.253 01 t=8E96E0 c=UNK key=P8 (13007002)
  ThreadId: 0000000a
  FunctionName: com.ibm.ws.security.core.distSecurityComponentImpl
  SourceId: com.ibm.ws.security.core.distSecurityComponentImpl
  Category: INFO
  ExtendedMessage: BBOO0222I: SECJ0212I: WCCM JAAS configuration information
         successfully pushed to login provider clas
Trace: 2005/05/06 17:27:42.254 01 t=8E96E0 c=UNK key=P8 (0000000A)
  Description: Log Boss/390 Error
  from filename: ./bborjtr.cpp
  at line: 932
  error message: BBOO0222I: SECJ0212I: WCCM JAAS configuration information
          successfully pushed to login provider class.
Trace: 2005/05/06 17:27:42.306 01 t=8E96E0 c=UNK key=P8 (13007002)
  ThreadId: 0000000a
  FunctionName: com.ibm.ws.security.core.distSecurityComponentImpl
  SourceId: com.ibm.ws.security.core.distSecurityComponentImpl
  Category: INFO
  ExtendedMessage: BBOO0222I: SECJ0240I: Security service initialization
         completed successfully
Trace: 2005/05/06 17:27:42.306 01 t=8E96E0 c=UNK key=P8 (0000000A)
  Description: Log Boss/390 Error
  from filename: ./bborjtr.cpp
  at line: 932
  error message: BBOO0222I: SECJ0240I: Security service initialization
          completed successfully
Trace: 2005/05/06 17:27:42.952 01 t=8E96E0 c=UNK key=P8 (13007002)
  ThreadId: 0000000a
  FunctionName: com.ibm.ws.objectpool.ObjectPoolService
  SourceId: com.ibm.ws.objectpool.ObjectPoolService
  Category: INFO
  ExtendedMessage: BBOO0222I: OBPL0007I: Object Pool Manager service
         is disabled.
Trace: 2005/05/06 17:27:53.512 01 t=8E96E0 c=UNK key=P8 (13007002)
  ThreadId: 0000000a
  FunctionName: com.ibm.ws.security.registry.UserRegistryImpl
  SourceId: com.ibm.ws.security.registry.UserRegistryImpl
  Category: AUDIT
  ExtendedMessage: BBOO0222I: SECJ0136I: Custom
         Registry:com.ibm.ws.security.registry.zOS.SAFRegistryImpl
         has been init
Trace: 2005/05/06 17:27:55.229 01 t=8E96E0 c=UNK key=P8 (13007002)
  ThreadId: 0000000a
  FunctionName: com.ibm.ws.security.role.PluggableAuthorizationTableProxy
  SourceId: com.ibm.ws.security.role.PluggableAuthorizationTableProxy
  Category: AUDIT
  ExtendedMessage: BBOO0222I: SECJ0157I: Loaded Vendor
         AuthorizationTable: com.ibm.ws.security.core.SAFAuthorizationTab
Trace: 2005/05/06 17:27:56.481 01 t=8E96E0 c=UNK key=P8 (13007002)
  ThreadId: 0000000a
  FunctionName: com.ibm.ws.security.core.distSecurityComponentImpl
  SourceId: com.ibm.ws.security.core.distSecurityComponentImpl
  Category: INFO
```

```
   ExtendedMessage: BBOO0222I: SECJ0243I: Security service started successfully
 Trace: 2005/05/06 17:27:56.481 01 t=8E96E0 c=UNK key=P8 (0000000A)
   Description: Log Boss/390 Error
   from filename: ./bborjtr.cpp
   at line: 932
   error message: BBOO0222I: SECJ0243I: Security service started successfully
 Trace: 2005/05/06 17:27:56.482 01 t=8E96E0 c=UNK key=P8 (13007002)
   ThreadId: 0000000a
   FunctionName: com.ibm.ws.security.core.distSecurityComponentImpl
   SourceId: com.ibm.ws.security.core.distSecurityComponentImpl
   Category: INFO
   ExtendedMessage: BBOO0222I: SECJ0210I: Security enabled true
 Trace: 2005/05/06 17:27:56.483 01 t=8E96E0 c=UNK key=P8 (0000000A)
   Description: Log Boss/390 Error
   from filename: ./bborjtr.cpp
   at line: 932
   error message: BBOO0222I: SECJ0210I: Security enabled true
```

Is there a stack trace or exception printed in the system log file?

A single stack trace tells a lot about the problem. What code initiated the code that failed? What is the failing component? Which class did the failure actually come from? Sometimes the stack trace is all that is needed to solve the problem and it can pinpoint the root cause. Other times, it can only give us a clue, and can actually be misleading. When support analyzes a stack trace, they can request additional trace if it is not clear what the problem is. If it seems to be security-related and the solution cannot be determined from the stack trace or problem description, you are asked to gather the following trace specification:
`SASRas=all=enabled:com.ibm.ws.security.*=all=enabled` from all processes involved.

Is this a distributed security problem or a local security problem?

- If the problem is local, that is the code involved does not make a remote method invocation, then troubleshooting is isolated to a single process. It is important to know when a problem is local versus distributed because the behavior of the object request broker (ORB), among other components, is different between the two. When a remote method invocation takes place, an entirely different security code path is entered.
- When you know that the problem involves two or more servers, the techniques of troubleshooting change. You need to trace all the servers involved simultaneously so that the trace shows the client and server sides of the problem. Make sure the timestamps on all machines match as closely as possible so that you can find the request and reply pair from two different processes. Enable both Secure Authentication Services (SAS) or z/SAS and Security trace using the trace specification: `SASRas=all=enabled:com.ibm.ws.security.*=all=enabled`.

  For more information on enabling trace, see the Tracing and logging configuration article.

  For more information on enabling trace, see Working with Trace.

Is the problem related to authentication or authorization?

Most security problems fall under one of these two categories. Authentication is the process of determining who the caller is. Authorization is the process of validating that the caller has the proper authority to invoke the requested method. When authentication fails, typically this failure is related to either the authentication protocol, authentication mechanism or user registry. When authorization fails, this is usually related to the application bindings from assembly and deployment and to the caller's identity who is accessing the method and the roles that are required by the method.

Is this a web or EJB request?

Web requests have a completely different code path than Enterprise JavaBeans (EJB) requests. Different security features exist for web requests than for EJB requests, requiring a completely different body of knowledge to resolve. For example, when using the Lightweight Third-Party Authentication (LTPA) authentication mechanism, the single sign-on feature (SSO) is available for web requests but not for EJB requests. Web requests involve HTTP header information that is not required by EJB requests due to the protocol differences. Also, the web container or servlet engine is involved in the entire process. Any of these components can be involved in the problem and all require consideration during troubleshooting, based on the type of request and where the failure occurs.

Secure EJB requests heavily involve the ORB and Naming components since they flow over the RMI/IIOP protocol. In addition, when Workload Manager (WLM) is enabled, other behavior

changes in the code can be observed. All of these components interact closely for security to work properly in this environment. At times, trace in any or all of these components might be necessary to troubleshoot problems in this area.

The trace specification to begin with is `SASRas=all=enabled:com.ibm.ws.security.*=all=enabled`. ORB trace is also very beneficial when the SAS/Security trace does not seem to pinpoint the problem.

Does the problem seem to be related to the Secure Sockets Layer (SSL)?

SSL is a totally distinct separate layer of security. Troubleshooting SSL problems is usually separate from troubleshooting authentication and authorization problems, and you have many considerations. Usually, SSL problems are first-time setup problems because the configuration can be difficult. Each client must contain the signer certificate of the server. During mutual authentication, each server must contain the client's signer certificate. Also, there can be protocol differences (SSLv3 vs. Transport Layer Security (TLS)), and listener port problems related to stale Interoperable Object References (IORs), that is IORs from a server, that reflect the port prior to the server restarting.

For SSL problems, sometimes you get a request for an SSL trace to determine what is happening with the SSL handshake. The SSL handshake is the process that occurs when a client opens a socket to a server. If anything goes wrong with the key exchange, cipher exchange, and so on, the handshake fails and the socket is not valid. Tracing JSSE (the SSL implementation that is used in WebSphere Application Server) involves the following steps:

- Set the following system property on the client and server processes: `-Djavax.net.debug=true`. For the server, add the system property to the generic JVM arguments property of the JVM settings page. For more information on this task, refer to Java virtual machine settings section of the *Administering applications and their environment* PDF book.
- Turn on ORB trace as well.
- Recreate the problem.

  The `SystemOut.log` of both processes contain the JSSE trace. You can find trace similar to the following example:

```
SSLConnection: install <com.ibm.sslite.e@3ae78375>
>> handleHandshakeV2 <com.ibm.sslite.e@3ae78375>
>> handshakeV2 type = 1
>> clientHello: SSLv2.
SSL client version: 3.0
...
...
...
JSSEContext: handleSession[Socket[addr=null,port=0,localport=0]]

<< sendServerHello.
SSL version: 3.0
SSL_RSA_WITH_RC4_128_MD5
HelloRandom
...
...
...
<< sendCertificate.
<< sendServerHelloDone.
>> handleData <com.ibm.sslite.e@3ae78375>
>> handleHandshake <com.ibm.sslite.e@3ae78375>
>> handshakeV3 type = 16

>> clientKeyExchange.
>> handleData <com.ibm.sslite.e@3ae78375>
>> handleChangeCipherSpec <com.ibm.sslite.e@3ae78375>
>> handleData <com.ibm.sslite.e@3ae78375>
>> handleHandshake <com.ibm.sslite.e@3ae78375>
>> handshakeV3 type = 20
>> finished.
<< sendChangeCipherSpec.
<< sendFinished.
```

Trace security

The classes that implement WebSphere Application Server security are:

- com.ibm.ws.security.*
- com.ibm.websphere.security.*

- com.ibm.WebSphereSecurityImpl.*
- SASRas
- com.ibm.ws.wim.* for tracing with a Virtual Member Manager (VMM) repository

To view detailed information on the run time behavior of security, enable trace on the following components and review the output:

- `com.ibm.ws.security.*=all=enabled:com.ibm.WebSphereSecurityImpl.*=all=enabled:com.ibm.websphere.security.*=all=enabled`. This trace statement collects the trace for the security runtime.
- `com.ibm.ws.console.security.*=all=enabled`. This trace statement collects the trace for the security center administrative console.
- `SASRas=all=enabled`. This trace statement collects the trace for SAS (low-level authentication logic).
- `com.ibm.ws.wim.*=all=enabled:com.ibm.websphere.wim.*=all=enabled`. This trace statement collects the trace for VMM.

Fine tuning SAS traces:

If a subset of classes need to be traced for the SAS/CSIv2 component, a system property can be specified with the class names comma separated:
`com.ibm.CORBA.securityTraceFilter=SecurityConnectionInterceptorImpl, VaultImpl, ...`

Fine tuning Security traces:

If a subset of packages need to be traced, specify a trace specification more detailed than `com.ibm.ws.security.*=all=enabled`. For example, to trace just dynamic policy code, you can specify `com.ibm.ws.security.policy.*=all=enabled`. To disable dynamic policy trace, you can specify `com.ibm.ws.security.policy.*=all=disabled`.

Configuring CSIv2, or SAS Trace Settings

Situations arise where reviewing trace for the CSIv2 or SAS authentication protocols can assist in troubleshooting difficult problems. This section describes how to enable to CSIv2 and SAS trace.

Enabling Client-Side CSIv2 and SAS Trace

To enable CSIv2 and SAS trace on a pure client, the following steps need to be taken:

- Edit the file `TraceSettings.properties` in the `/WebSphere/AppServer/properties` directory.
- In this file, change `traceFileName=` to point to the path in which you want the ouput file created. Make sure you put a double backslash (\\) between each subdirectory. For example, `traceFileName=c:\\WebSphere\\AppServer\\logs\\sas_client.log`
- In this file, add the trace specification string: `SASRas=all=enabled`. Any additional trace strings can be added on separate lines.
- Point to this file from within your client application. On the Java command line where you launch the client, add the following system property:
  `-DtraceSettingsFile=TraceSettings.properties`.

  Note: Do not give the fully qualified path to the `TraceSettings.properties` file. Make sure that the `TraceSettings.properties` file is in your class path.

Enabling Server-Side CSIv2 and SAS Trace

To enable SAS trace in an application server, complete the following:

- Add the trace specification, `SASRas=all=enabled`, to the `server.xml` file or add it to the Trace settings within the WebConsole GUI.
- Typically it is best to also trace the authorization security runtime in addition to the authentication protocol runtime. To do this, use the following two trace specifications in combination: `SASRas=all=enabled:com.ibm.ws.security.*=all=enabled`.
- When troubleshooting a connection type problem, it is beneficial to trace both CSIv2 and SAS or CSIv2 and z/SAS and the ORB. To do this, use the following three trace specifications:
  `SASRas=all=enabled:com.ibm.ws.security.*=all=enabled:ORBRas=all=enabled`.
- In addition to adding these trace specifications, for ORB trace there are a couple of system properties that also need to be set. Go to the ORB settings in the GUI and add the following two properties: `com.ibm.CORBA.Debug=true` and `com.ibm.CORBA.CommTrace=true`.

CSIv2 CORBA minor codes

Whenever exceptions occur within the security code on either the client or server, the eventual exception becomes a Common Object Request Broker Architecture (CORBA) exception. Any exception that occurs gets embedded in a CORBA exception because the CORBA architecture is used by the security service for its own inter-process communication. CORBA exceptions are generic and indicate a problem in communication between two components. CORBA minor codes are more specific and indicate the underlying reason that a component could not complete a request.

The following shows the CORBA minor codes that a client can expect to receive after running a security-related request such as authentication. It also includes the CORBA exception type that the minor code appears in.

The following exception shows an example of a CORBA exception where the minor code is 49424300 and indicates Authentication Failure. Typically, a descriptive message is also included in the exception to assist in troubleshooting the problem. Here, the detailed message is: “Exception caught invoking authenticateBasicAuthData from SecurityServer for user jdoe. Reason: com.ibm.WebSphereSecurity.AuthenticationFailedException” which indicates that the authentication failed for user *jdoe*.

The completed field in the exception indicates whether the method was completed or not. In the case of a NO_PERMISSION, never invoke the message; therefore it is always `completed:No`. Other exceptions that are caught on the server side can have a completed status of “Maybe” or “Yes”.

```
org.omg.CORBA.NO_PERMISSION: Caught WSSecurityContextException in
WSSecurityContext.acceptSecContext(),
reason: Major Code[0] Minor Code[0] Message[Exception caught invoking
authenticateBasicAuthData from SecurityServer for user jdoe.  Reason:
com.ibm.WebSphereSecurity.AuthenticationFailedException]  minor code: 49424300
completed: No

at com.ibm.ISecurityLocalObjectBaseL13Impl.PrincipalAuthFailReason.
map_auth_fail_to_minor_code(PrincipalAuthFailReason.java:83)
        at com.ibm.ISecurityLocalObjectBaseL13Impl.CSIServerRI.receive_request
               (CSIServerRI.java:1569)
        at com.ibm.rmi.pi.InterceptorManager.iterateReceiveRequest
              (InterceptorManager.java:739)
        at com.ibm.CORBA.iiop.ServerDelegate.dispatch(ServerDelegate.java:398)
        at com.ibm.rmi.iiop.ORB.process(ORB.java:313)
        at com.ibm.CORBA.iiop.ORB.process(ORB.java:1581)
        at com.ibm.rmi.iiop.GIOPConnection.doWork(GIOPConnection.java:1827)
        at com.ibm.rmi.iiop.WorkUnitImpl.doWork(WorkUnitImpl.java:81)
        at com.ibm.ejs.oa.pool.PooledThread.run(ThreadPool.java:91)
        at com.ibm.ws.util.CachedThread.run(ThreadPool.java:149)
```

Table 166. CORBA minor codes after running a security-related request such as authentication. The following table shows the CORBA minor codes which a client can expect to receive after running a security-related request such as authentication. The client can be either a stand-alone client or a server acting as a client. It also includes the CORBA exception type that the minor code would appear in.

Minor code name	Minor code value (in hex)	Exception type (all in the package of org.omg.CORBA.*)	Minor code description	Retry performed by stand-alone client (when authenticationRetryEnabled = true)	Retry performed by server acting as a client (when authenticationRetryEnabled = true)
AuthenticationFailed	49424300	NO_PERMISSION	This code is a generic authentication failed error. It does not give any details about whether or not the user ID or password is valid. Some user registries can choose to use this type of error code, others can choose to use the next three types that are more specific.	Yes	Yes

Table 166. CORBA minor codes after running a security-related request such as authentication (continued). The following table shows the CORBA minor codes which a client can expect to receive after running a security-related request such as authentication. The client can be either a stand-alone client or a server acting as a client. It also includes the CORBA exception type that the minor code would appear in.

Minor code name	Minor code value (in hex)	Exception type (all in the package of org.omg.CORBA.*)	Minor code description	Retry performed by stand-alone client (when authenticationRetryEnabled = true)	Retry performed by server acting as a client (when authenticationRetryEnabled = true)
InterceptLocateException	494210B8	INTERNAL	This indicates a problem when processing an incoming locate request.	No	No
InvalidUserid	49424301	NO_PERMISSION	This code occurs when the registry returns bad user ID.	Yes	No
InvalidPassword	49424302	NO_PERMISSION	This code occurs when the registry returns a bad password.	Yes	No
InvalidSecurityCredentials	49424303	NO_PERMISSION	This is a generic error indicating that the credentials are bad for some reason. It might be that the right attributes are not set.	Yes, if client has BasicAuth credential (token based credential was rejected in the first place).	Yes
InvalidRealm	49424304	NO_PERMISSION	This code occurs when the REALM in the token received from the client does not match the server's current realm.	No	No
ValidationFailed	49424305	NO_PERMISSION	A validation failure occurs when a token is sent from the client or server to a target server but the token format or the expiration is not valid.	Yes, if client has BasicAuth credential (token based credential was rejected in the first place).	Yes
CredentialTokenExpired	49424306	NO_PERMISSION	This code is more specific about why the validation failed. In this case, the token has an absolute lifetime and the lifetime has expired. Therefore, it is no longer a valid token and cannot be used.	Yes, if client has BasicAuth credential (token based credential was rejected in the first place).	Yes
InvalidCredentialToken	49424307	NO_PERMISSION	This is more specific about why the validation failed. In this case, the token cannot be decrypted or the data within the token is not readable.	Yes, if client has BasicAuth credential (token based credential was rejected in the first place).	No
SessionDoesNotExist	49424308	NO_PERMISSION	This indicates that the CSIv2 session does not exist on the server. Typically, a retry occurs automatically and successfully creates a new session.	Yes	Yes

Table 166. CORBA minor codes after running a security-related request such as authentication (continued). The following table shows the CORBA minor codes which a client can expect to receive after running a security-related request such as authentication. The client can be either a stand-alone client or a server acting as a client. It also includes the CORBA exception type that the minor code would appear in.

Minor code name	Minor code value (in hex)	Exception type (all in the package of org.omg.CORBA.*)	Minor code description	Retry performed by stand-alone client (when authenticationRetryEnabled = true)	Retry performed by server acting as a client (when authenticationRetryEnabled = true)
SessionConflictingEvidence	49424309	NO_PERMISSION	This indicates that a session already exists on the server that matches the context_id sent over by the client. However, the information provided by the client for this EstablishContext message is different from the information originally provided to establish the session.	Yes	Yes
SessionRejected	4942430A	NO_PERMISSION	This indicates that the session referenced by the client has been previously rejected by the server.	Yes	Yes
SecurityServerNotAvailable	4942430B	NO_PERMISSION	This error occurs when the server cannot contact the local or remote security server in order to authenticate or validate.	No	No
InvalidIdentityToken	4942430C	NO_PERMISSION	This error indicates that identity cannot be obtained from the identity token when Identity Assertion is enabled.	No	No
IdentityServerNotTrusted	4942430D	NO_PERMISSION	This indicates that the server ID of the sending server is not on the target server's trusted principal list.	No	No
InvalidMessage	4942430E	NO_PERMISSION	This indicates that the CSIv2 message format is not valid for the receiving server.	No	No
MappingFailed	4942430F	NO_PERMISSION	This indicates an error occurred mapping an inbound subject using the RMI Inbound system login configuration.	No	No
RevokedSecurityName	49424310	NO_PERMISSION	This indicates that the user id is revoked.	Yes	No
ExpiredPassword	49424311	NO_PERMISSION	This indicates that the password is expired.	Yes	No
AuthenticationNotSupported	49421090	NO_PERMISSION	This error occurs when a mechanism does not support authentication (very rare).	No	No
InvalidSecurityMechanism	49421091	NO_PERMISSION	This is used to indicate that the specified security mechanism is not known.	No	No
CredentialNotAvailable	49421092	NO_PERMISSION	This indicates a credential is not available when it is required.	No	No

Table 166. CORBA minor codes after running a security-related request such as authentication (continued). The following table shows the CORBA minor codes which a client can expect to receive after running a security-related request such as authentication. The client can be either a stand-alone client or a server acting as a client. It also includes the CORBA exception type that the minor code would appear in.

Minor code name	Minor code value (in hex)	Exception type (all in the package of org.omg.CORBA .*)	Minor code description	Retry performed by stand-alone client (when authenticationRetryEnabled = true)	Retry performed by server acting as a client (when authenticationRetryEnabled = true)
SecurityMechanismNotSupported	49421093	NO_PERMISSION	This error occurs when a security mechanism that is specified in the CSIv2 token is not implemented on the server.	No	No
ValidationNotSupported	49421094	NO_PERMISSION	This error occurs when a mechanism does not support validation, such as LocalOS. This error does not occur since the LocalOS credential is not a forwardable credential, therefore, validation never needs to be called on this credential.	No	No
CredentialTokenNotSet	49421095	NO_PERMISSION	This is used to indicate that the token inside the credential is null.	No	No
InvalidEvidence	49421096	NO_PERMISSION	This error indicates that client authentication is required at the server. However, authentication information is not present in the method request from the client.	No	No
UserRegistryMethod_Protected	49421098	NO_PERMISSION	This error indicates that an attempt was made to remotely access a protected UserRegistry method.	No	No
ServerConnectionFailed	494210A0	COMM_FAILURE	This error is used when a connection attempt fails.	Yes (via ORB retry)	Yes (via ORB retry)
CorbaSystemException	494210B0	INTERNAL	This code is a generic CORBA specific exception in system code.	No	No
JavaException	494210B1	INTERNAL	This is a generic error that indicated that an unexpected Java exception occurred.	No	No
ValueIsNull	494210B2	INTERNAL	This code is used to indicate that a value or parameter that passed in is null.	No	No
EffectivePolicyNotPresent	494210B3	INTERNAL	This indicates that an effective policy object for CSIv2 is not present. This object is used to determine what security configuration features are specified.	No	No
NullPointerException	494210B4	INTERNAL	This code is used to indicate that a NullPointerException is caught in the runtime.	No	No

Table 166. CORBA minor codes after running a security-related request such as authentication (continued). The following table shows the CORBA minor codes which a client can expect to receive after running a security-related request such as authentication. The client can be either a stand-alone client or a server acting as a client. It also includes the CORBA exception type that the minor code would appear in.

Minor code name	Minor code value (in hex)	Exception type (all in the package of org.omg.CORBA.*)	Minor code description	Retry performed by stand-alone client (when authenticationRetryEnabled = true)	Retry performed by server acting as a client (when authenticationRetryEnabled = true)
ErrorGettingClassInstance	494210B5	INTERNAL	This indicates a problem loading a class dynamically.	No	No
MalFormedParameters	494210B6	INTERNAL	This indicates parameters are not valid.	No	No
DuplicateSecurityAttributeType	494210B7	INTERNAL	This indicates a duplicate credential attribute that is specified during the set_attributes operation.	No	No
MethodNotImplemented	494210C0	NO_IMPLEMENT	This indicates that a method invoked is not implemented.	No	No
GSSFormatError	494210C5	BAD_PARAM	This code indicates that a Generic Security Services (GSS) encoding or decoding routine has created an exception.	No	No
TagComponentFormatError	494210C6	BAD_PARAM	This code indicates that a tag component cannot be read properly.	No	No
InvalidSecurityAttributeType	494210C7	BAD_PARAM	This code indicates an attribute type specified during the set_attributes operation is not a valid type.	No	No
SecurityConfigError	494210CA	INITIALIZE	This code indicates a problem exists between the client and server configuration.	No	No

For current information available from IBM Support on known problems and their resolution, see the IBM Support page.

IBM Support has documents that can save you time gathering information needed to resolve this problem. Before opening a PMR, see the IBM Support page.

Security configuration and enablement errors

Use this information to troubleshoot problems with configuring or enabling security.

Note: This topic references one or more of the application server log files. As a recommended alternative, you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log` , `SystemErr.log`, `trace.log`, and `activity.log` files on distributed and IBM i systems. You can also use HPEL in conjunction with your native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

What kind of error are you seeing?

- “LTPA password not set. validation failed error message”
- “The setupClient.bat or setupClient.sh file is not working correctly”
- HP-UX “Java HotSpot Server VM warning”
- “WebSphere Application Server Version 6 is not working correctly with Enterprise Workload Manager (EWLM)” on page 1003
- If you successfully configured security, but are now having problems accessing web resources or the administrative console, refer to Errors or access problems after enabling security.
- “NMSV0610I: A NamingException is being thrown from a javax.naming.Context implementation” on page 1003
- “Performance servlet displays authorization errors and cannot provide statistics” on page 1003
- “Name value is invalid displays when migrating users and groups after the JACC provider for Tivoli is configured” on page 1004
- “Sun JDK can not read a PKCS12 keystore created by the Application Server” on page 1004
- “Calling a secure resource from a non-secure resource is not supported” on page 1004

For general tips on diagnosing and resolving security-related problems, see the topic Troubleshooting the security component.

“LTPA password not set. validation failed” error message

“LTPA password not set. validation failed” message displayed as error in the administrative console after saving administrative or application security settings

This error can be caused if, when configuring WebSphere Application Server security, LTPA is selected as the authentication mechanism and the LTPA password field is not set. To resolve this problem:
- Select **Security** > **Global security** > **Authentication mechanisms and expiration** > **LTPA**.
- Complete the password and confirm password fields.
- Click OK.
- Try setting administrative or application security again.

The setupClient.bat or setupClient.sh file is not working correctly

The `setupClient.bat` file on Windows operating systems and the `setupClient.sh` file on Linux and UNIX-based platforms incorrectly specify the location of the SOAP security properties file.

Windows In the `setupClient.bat` file, the correct location is:

```
set CLIENTSOAP=-Dcom.ibm.SOAP.ConfigURL=file:%WAS_HOME%/properties/soap.client.props
```

AIX Linux AIX HP-UX Solaris In the `setupClient.sh` file, the CLIENTSOAP variable is:

```
CLIENTSOAP=-Dcom.ibm.SOAP.ConfigURL=file:$WAS_HOME/properties/soap.client.props
```

In the `setupClient.bat` and the `setupClient.sh` files, complete the following steps:
1. Remove the leading slash (/) after `file:`.
2. Change `sas` to `soap`.

HP-UX
Java HotSpot Server VM warning

After you enable security on HP-UX 11i platforms, the following error in the `native_stdout.log` file occurs, along with a core dump and WebSphere Application Server does not start:

```
Java HotSpot(TM) Server VM warning:
Unexpected Signal 11 occurred under user-defined signal handler 0x7895710a
```

To work around this error, apply the fixes recommended by Hewlett Packard for Java at the following URL: http://www.hp.com/products1/unix/java/infolibrary/patches.html.

WebSphere Application Server Version 6 is not working correctly with Enterprise Workload Manager™ (EWLM)

To use WebSphere Application Server Version 6 with EWLM, you must manually update the WebSphere Application Server server.policy files. For example:

```
grant codeBase "file:/<EWLM_Install_Home>/classes/ARM/arm4.jar" {
 permission java.security.AllPermission;
};
```

Otherwise, you might encounter a Java 2 security exception for violating the Java 2 security permission.

For more information on configuring server.policy files, refer to the server.policy file permissions section in the *Developing and deploying applications* PDF book.

For current information available from IBM Support on known problems and their resolution, see the IBM Support page.

IBM Support has documents that can save you time gathering information needed to resolve this problem. Before opening a PMR, see the IBM Support page.

NMSV0610I: A NamingException is being thrown from a javax.naming.Context implementation

If you use CSIv2 inbound authentication, basic authentication is required, and Java clients running with com.ibm.CORBA.validateBasicAuth=true might fail with the following exception:

```
If you use CSIv2 inbound authentication, basic authentication is required, and Java™
clients running with com.ibm.CORBA.validateBasicAuth=true might fail with the
following exception:

NMSV0610I: A NamingException is being thrown from a javax.naming.Context
           implementation. Details follow:

Context implementation: com.ibm.ws.naming.jndicos.CNContextImpl
Context method: lookupExt
Context name: TestaburgerNode01Cell/nodes/TestaburgerNode01/servers/server1
Target name: SecurityServer
Other data: "" ""
Exception stack trace: javax.naming.NoPermissionException: NO_PERMISSION
exception caught. Root exception is org.omg.CORBA.NO_PERMISSION:
vmcid: 0x49421000 minor code: 92 completed: No
...
SECJ0395E: Could not locate the SecurityServer at host/port:9.42.72.27/9100
to validate the userid and password entered. You may need to specify valid
securityServerHost/Port in (WAS_INSTALL_ROOT)/properties/sas.client.props file.
```

To fix this problem, modify the `com.ibm.CORBA.validateBasicAuth=false` property in the client's `sas.clients.props` file, which is located in `WAS_HOME/profiles/<profile-name>/properties`, and then run the client.

Performance servlet displays authorization errors and cannot provide statistics

In WebSphere Application Server Version 6.1, when administrative security is enabled, the administration service is locked down. However, if application security is not enabled, an authentication challenge does not occur for incoming requests and, consequently, credentials do not exist for the performance servlet to access the administration service.

If administrative security is enabled, you also must enable application security for the performance servlet to process incoming requests.

“Name value is invalid” displays when migrating users and groups after the JACC provider for Tivoli is configured

When you use the migrateEAR utility to migrate the changes that were made to console users and groups after the JACC provider for Tivoli Access Manager is configured, the following configuration error displays in the systemOut.log file.

```
<specialSubjects> name value is invalid
```

The migrateEAR utility migrates the user and group data that is contained in the admin-authz.xml file. However, the migrateEAR utility does not convert the XML tags that are listed in the admin-authz.xml file if the pdwas-admin group is added to the administrator access control list (ACL) in Tivoli Access Manager prior to migration.

To resolve this error, enter the following command in padadmin to check whether the pdwas-admin group is in the administrator ACL before you migrate:

```
acl show
_WebAppServer_deployedResources_Roles_administrator_admin-authz_ACL
```

The following result should display:

```
ACL Name:
_WebAppServer_deployedResources_Roles_administrator_admin-authz_ACL
Description: Created by the Tivoli Access Manager
for Websphere Application Server Migration Tool.
Entries:
User sec_master TcmdbsvaBR1
Group pdwas-admin T[WebAppServer]i
```

If the pdwas-admin group is not listed, then enter the following command in pdadmin to modify the ACL to add the pdwas-admin group:

```
acl modify
_WebAppServer_deployedResources_Roles_administrator_admin
-authz_ACL set gruop pdwas-admin T [WebAppServer]i
```

Sun JDK can not read a PKCS12 keystore created by the Application Server

A Sun JDK is not able to read a PKCS12 keystore created by the Application Server. The reason for this is that the PKCS12 implementation used by the IBM SDK and the Application Server is different than the implementation used by the Sun JDK. The difference causes problems when a Sun JDK is used to read the default trustore, trust.p12, or keystore, key.P12 created by the Application Server.

Because the truststore can not be read by the Sun JDK, you must first extract the certificates from the trustore using an IBM SDK. You can then import these certificates into a keystore that the Sun JDK can recognize correctly, such as a JKS keystore.

Calling a secure resource from a non-secure resource is not supported

If you have a non-secure resource (such as a JSP or a servlet) that calls a secure resource, the application might fail if the non-secure resource collects data from users and then posts this data to secure JSP or servlet files for processing.

To avoid this situation, structure your web application so that users are forced to login before the application performs any HTTP POST actions to the secure JSP or servlet files. To accomplish this, secure the first resource using whatever security mechanism that you choose (such as basic auth, form-login or cert).

This restriction is because basic auth and form-login use the servlet sendRedirect method, which has several implications for the user. The sendRedirect method is used twice during basic auth and form-login.

The sendRedirect method initially displays the basic auth or form-login page in the web browser. It later redirects the web browser back to the originally requested protected page. The sendRedirect(String URL) method tells the web browser to use the HTTP GET request to access the page that is specified in the web address. If HTTP POST is the first request to a protected JSP or servlet file, and no previous authentication or login occurred, then HTTP POST is not delivered to the requested page. However, HTTP GET is delivered because basic auth and form-login use the sendRedirect method, which behaves as an HTTP GET request which attempts to display a requested page after a login occurs.

Security enablement followed by errors

Use this information if you are experiencing errors after security is enabled.

Note: This topic references one or more of the application server log files. As a recommended alternative, you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log` , `SystemErr.log`, `trace.log`, and `activity.log` files on distributed and IBM i systems. You can also use HPEL in conjunction with your native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

What kind of error are you seeing?

- Authentication error accessing a web page
- Authorization error accessing a web page
- “Authentication fails when code pages differ between the client and the server” on page 1006
- Error Message: `CWSCJ0314E: Current Java 2 security policy reported a potential violation`
- `CWMSG0508E: The JMS Server security service was unable to authenticate user ID:` error displayed in SystemOut.log when starting an application server
- Error Message: `CWSCJ0237E: One or more vital LTPAServerObject configuration attributes are null or not available` after enabling security and starting the application server
- An AccessControlException is reported in the SystemOut.log
- Error Message: `CWSCJ0336E: Authentication failed for user {0} because of the following exception {1}`
- “Error Message: An unexpected exception occurred initializing the security collaborator.java.lang.SecurityException: AuthConfigFactory error: java.lang.ClassNotFoundException: org.apache.geronimo.components.jaspi.AuthConfigFactoryImpl” on page 1010
- “Error Message: SECJ8032W: AuthConfigFactory is undefined, using the default JASPI factory implementation class” on page 1011
- “Error Message: SECJ0352E: Could not get the users matching the pattern {0} because of the following exception {1}” on page 1011
- “Validation of LTPA token failed due to invalid keys or token type” on page 1011
- “Generate keys error when using the Profile Management tool to create a new profile” on page 1012
- “Some security roles are not immediately available for a secured application where LDAP has Tivoli Access Manager enabled” on page 1012
- “After setting domain realm to not trusted, global security settings cannot be used” on page 1013
- “Updated global security realm names are duplicated” on page 1013
- “Errors may occur when the session security feature is turned on” on page 1013
-

For general tips on diagnosing and resolving security-related problems, see the topic Troubleshooting the security component.

IBM Support has documents and tools that can save you time gathering information needed to resolve problems. Before opening a problem report, see the Support page:

- http://www.ibm.com/software/webservers/appserv/was/support/

Authentication error accessing a Web page

Possible causes for authentication errors include:

- Incorrect user name or passwords. Check the user name and password and make sure that they are correct.
- Security configuration error : User registry type is not set correctly. Check the user registry property in administrative security settings in the administrative console. Verify that the user registry property is the intended user registry.
- Internal program error. If the client application is a Java stand-alone program, this program might not gather or send credential information correctly.

If the user registry configuration, user ID, and password appear correct, use the WebSphere Application Server trace to determine the cause of the problem. To enable security trace, use the **com.ibm.ws.security.*=all=enabled** trace specification.

Authorization error accessing a Web page

If a user who is supposed to have access to a resource does not, a configuration step is probably missing. For more information on configuring access to resources, review the chapter Authorizing access to administrative roles in the *Securing applications and their environment* PDF book.

Specifically:

- Check the required roles for the accessed web resource.
- Check the authorization table to make sure that the user, or the groups to which the user belongs, is assigned to one of the required roles.
- View required roles for the web resource in the deployment descriptor of the web resource.
- View the authorization table for the application that contains the web resource, using the administrative console.
- Test with a user who is granted the required roles, to see if the user can access the problem resources.
- If the user is required to have one or more of the required roles, use the administrative console to assign that user to required roles, stop, and restart the application.

If the user is granted required roles, but still fails to access the secured resources, enable security trace, using **com.ibm.ws.security.*=all=enabled** as the trace specification. Collect trace information for further resolution.

Authentication fails when code pages differ between the client and the server

When a client uses a code page that is different from the server, and non-US-ASCII characters are used for the user ID and password during basic authentication, the login does not succeed. The HTTP header does not include the encoding method information that is necessary to translate the encoded data, so the server does not know how to decode the information correctly.

Use a login form that relies on POST parameters, which are in the HTML body text. The encoding for the text is sent by the browser and so is capable of being decoded properly.

Note: Web services customers are not able to use form login to resolve this problem. Users must ensure there is consistency in the code pages between the client and the server.

Error Message: CWSCJ0314E: Current Java 2 security policy reported a potential violation on server

If you find errors on your server similar to:

```
Error Message: CWSCJ0314E: Current Java 2 Security policy reported a potential violation of
Java 2 Security Permission. Please refer to Problem Determination Guide for further information.
{0}Permission/:{1}Code/:{2}{3}Stack Trace/:{4}Code Base Location/:{5}
```

The Java security manager `checkPermission` method has reported a `SecurityException` exception .

The reported exception might be critical to the secure system. Turn on security trace to determine the potential code that might have violated the security policy. Once the violating code is determined, verify if the attempted operation is permitted with respect to Java 2 Security, by examining all applicable Java 2 security policy files and the application code.

A more detailed report is enabled by either configuring RAS trace into debug mode, or specifying a Java property.

- Check the Tracing and logging configuration article for instructions on how to configure Reliability Availability Serviceability (RAS) trace into debug mode, or
- Specify the following property in the **Application Servers** > ***server_name*** > **ProcessDefinition** > **Java Virtual Machine** panel from the administrative console in the Generic JVM arguments panel:
  - Add the **`java.security.debug`** run-time flag
  - Valid values:

    access
    Print all debug information including required permission, code, stack, and code base location.

    stack Print debug information including required permission, code, and stack.

    failure Print debug information including required permission, and code.

For a review of Java security policies, see the Java 2 Security documentation at http://java.sun.com/j2se/1.3/docs/guide/security/index.html.

Tip: If the application is running with a Java Mail application programming interface (API), this message might be benign. You can update the *`installed Enterprise Application root`*`/META-INF/was.policy` file to grant the following permissions to the application:

- `permission java.io.FilePermission "${user.home}${/}.mailcap", "read";`
- `permission java.io.FilePermission "${user.home}${/}.mime.types", "read";`
- `permission java.io.FilePermission "${java.home}${/}lib${/}mailcap", "read";`
- `permission java.io.FilePermission "${java.home}${/}lib${/}mime.types", "read";`

Error message: CWMSG0508E: The JMS Server security service was unable to authenticate user ID:" error displayed in SystemOut.log when starting an application server

This error can result from installing the Java Message Service (JMS) API sample and then enabling security. You can follow the instructions in the Configure and Run page of the corresponding JMS sample documentation to configure the sample to work with WebSphere Application Server security.

You can verify the installation of the message-driven bean sample by launching the installation program, selecting `Custom`, and browsing the components which are already installed in the Select the features you like to install panel. The JMS sample is shown as Message-Driven Bean Sample, under Embedded Messaging.

You can also verify this installation by using the administrative console to open the properties of the application server that contains the samples. Select `MDBSamples` and click uninstall.

Error message: CWSCJ0237E: One or more vital LTPAServerObject configuration attributes are null or not available after enabling security and starting the application server

This error message can result from selecting Lightweight Third Party Authentication (LTPA) as the authentication mechanism, but not generating the LTPA keys. The LTPA keys encrypt the LTPA token.

To resolve this problem:

1. Click **Security** > **Global security** > **Authentication** > **Authentication mechanisms and expiration** > **LTPA**.
2. Enter a password, which can be anything.
3. Enter the same password in Confirm Password.
4. Click Apply.
5. Click Generate Keys.
6. Click Save.

The AccessControlException exception, is reported in the SystemOut.log

The problem is related to the Java 2 security feature of WebSphere Application Server, the API-level security framework that is implemented in WebSphere Application Server. An exception similar to the following example displays. The error message and number can vary.

```
CWSRV0020E: [Servlet Error]-[validator]: Failed to load servlet:
java.security.AccessControlException: access denied
(java.io.FilePermission
app_server_root/systemApps/isclite.ear/isclite.war/WEB-INF/validation.xml read)
```

For an explanation of Java 2 security, how and why to enable or disable it, how it relates to policy files, and how to edit policy files, see the Java 2 security topic in the *Securing applications and their environment* PDF book.The topic explains that Java 2 security is not only used by this product, but developers can also implement it for their business applications. Administrators might need to involve developers, if this exception is created when a client tries to access a resource that is hosted by WebSphere Application Server.

Possible causes of these errors include:
- Syntax errors in a policy file.
- Syntax errors in permission specifications in the `ra.xml` file that is bundled in a `.rar` file. This case applies to resource adapters that support connector access to CICS® or other resources.
- An application is missing the specified permission in a policy file, or in permission specifications in `ra.xml` file bundled in a `.rar` file.
- The class path is not set correctly, preventing the permissions for the `resource.xml` file for Service Provider Programming Interface (SPI) from being correctly created.
- A library called by an application, or the application, is missing a doPrivileged block to support access to a resource.
- Permission is specified in the wrong policy file.

To resolve these problems:
- Check all of the related policy files to verify that the permission shown in the exception, for example java.io.FilePermission, is specified.
- Look for a related ParserException exception in the `SystemOut.log` file which reports the details of the syntax error.

  For example:

```
CWSCJ0189E: Caught ParserException while creating template for application policy

profile_root/config/cells/cell_name/nodes/node_name/app.policy
```

  Where:
  - *cell_name* represents the name of your cell.
  - *profile_name* represents the name of your profile.
  - *node_name* represents the name of your node.

  The exception is com.ibm.ws.security.util.ParserException: line 18: expected ';', found 'grant'
- Look for a message similar to: `CWSCJ0325W: The permission permission specified in the policy file is unresolved.`
- Check the call stack to determine which method does not have the permission. Identify the class path of this method. If it is hard to identify the method, enable the Java2 security Report.
  - Configuring RAS trace by specifying `com.ibm.ws.security.core.*=all=enabled`, or specifying a Java **property.java.security.debug** property. Valid values for the **java.security.debug** property are:

access
Print all debug information including: required permission, code, stack, and code base location.

stack Print debug information including: required permission, code, and stack.

failure Print debug information including: required permission and code.

– The report shows:

Permission
The missing permission.

Code Which method has the problem.

Stack Trace
Where the access violation occurred.

CodeBaseLocation
The detail of each stack frame.

Usually, permission and code are enough to identify the problem. The following example illustrates a report:

```
Permission:
app_server_root/logs/server1/SystemOut_02.08.20_11.19.53.log :
access denied (java.io.FilePermission
app_server_root/logs/server1/SystemOut_02.08.20_11.19.53.log delete)

Code:
 com.ibm.ejs.ras.RasTestHelper$7  in
{file:app_server_root/installedApps/app1/JrasFVTApp.ear/RasLib.jar
}

Stack Trace:

java.security.AccessControlException: access denied (java.io.FilePermission
app_server_root/logs/server1/SystemOut_02.08.20_11.19.53.log delete
)
        at java.security.AccessControlContext.checkPermission
                           (AccessControlContext.java(Compiled Code))
        at java.security.AccessController.checkPermission
                           (AccessController.java(Compiled Code))
        at java.lang.SecurityManager.checkPermission
                           (SecurityManager.java(Compiled Code))
                            .
Code Base Location:

com.ibm.ws.security.core.SecurityManager :
file:/app_server_root/plugins/com.ibm.ws.runtime_6.1.0.jar

  ClassLoader: com.ibm.ws.bootstrap.ExtClassLoader
  Permissions granted to CodeSource
(file:/app_server_root/plugins/com.ibm.ws.runtime_6.1.0.jar <no certificates>

  {
    (java.util.PropertyPermission java.vendor read);
    (java.util.PropertyPermission java.specification.version read);
    (java.util.PropertyPermission line.separator read);
    (java.util.PropertyPermission java.class.version read);
    (java.util.PropertyPermission java.specification.name read);
    (java.util.PropertyPermission java.vendor.url read);
    (java.util.PropertyPermission java.vm.version read);
    (java.util.PropertyPermission os.name read);
    (java.util.PropertyPermission os.arch read);
   }
   ( This list continues.)
```

Where:
- app1 represents the name of your application.
- *app_server_root* represents the installation root directory for WebSphere Application Server.
- *profile_root* represents the location and name of a particular profile in your system.
- profile1 or *profile_name* represents the name of your profile.
- server1 or *server_name*represents the name of your application server.

- If the method is SPI, check the `resources.xml` file to ensure that the class path is correct.
- To confirm that all of the policy files are loaded correctly, or what permission each class path is granted, enable the trace with **com.ibm.ws.security.policy.*=all=enabled**. All loaded permissions are listed in the `trace.log` file. Search for the `app.policy`, `was.policy` and `ra.xml` files. To check the permission list for a class path, search for Effective Policy for *classpath*.

- If there are any syntax errors in the policy file or the `ra.xml` file, correct them with the policy tool. Avoid editing the policy manually, because syntax errors can result. For additional information about using this tool, refer to the section Using PolicyTool to edit policy files in the *Developing and deploying applications* PDF book.
- If a permission is listed as Unresolved, it does not take effect. Verify that the specified permission name is correct.
- If the class path that is specified in the `resource.xml` file is not correct, correct it.
- If a required permission does not exist in either the policy files or the `ra.xml` file, examine the application code to see if you need to add this permission. If so, add it to the proper policy file or the `ra.xml` file.
- If the permission is not granted outside of the specific method that is accessing this resource, modify the code needs to use a doPrivileged block.
- If this permission does exist in a policy file or a `ra.xml` file and the permission was loaded correctly, but the class path still does not have the permission in its list, the location of the permission might not be correct. Read the Java 2 security chapter in the *Securing applications and their environment* PDF book carefully to determine in which policy file or `ra.xml` file to specify that permission.

Tip: If the application is running with the Java Mail API, you can update the *installed Enterprise Application root*`/META-INF/was.policy` file to grant the following permissions to the application:

- `permission java.io.FilePermission "${user.home}${/}.mailcap", "read";`
- `permission java.io.FilePermission "${user.home}${/}.mime.types", "read";`
- `permission java.io.FilePermission "${java.home}${/}lib${/}mailcap", "read";`
- `permission java.io.FilePermission "${java.home}${/}lib${/}mime.types", "read";`

Error Message: CWSCJ0336E: Authentication failed for user {0} because of the following exception {1}

This error message results if the user ID that is indicated is not found in the Lightweight Directory Access Protocol (LDAP) user registry. To resolve this problem:

1. Verify that your user ID and password are correct.
2. Verify that the user ID exists in the registry.
3. Verify that the base distinguished name (DN) is correct.
4. Verify that the user filter is correct.
5. Verify that the bind DN and the password for the bind DN are correct. If the bind DN and password are not specified, add the missing information and retry.
6. Verify that the host name and LDAP type are correct.

Consult with the administrator of the user registry if the problem persists.

Error Message: An unexpected exception occurred initializing the security collaborator.java.lang.SecurityException: AuthConfigFactory error: java.lang.ClassNotFoundException: org.apache.geronimo.components.jaspi.AuthConfigFactoryImpl

This error message occurs when your java.security file is missing an entry for the JASPI Provider. The default location for the java.security file is `install_dir/properties`. Edit the java.security file and add the following lines to it:.

```
#
# The fully qualified class name of the default JASPI factory implementation class.
#
authconfigprovider.factory=com.ibm.ws.security.jaspi.ProviderRegistry
```

Note: This error only appears if you explicitly set your configuration to use this class. Otherwise, you might see error message SECJ8032W below.

Error Message: SECJ8032W: AuthConfigFactory is undefined, using the default JASPI factory implementation class

This error message occurs if the JASPI factory implementation is not defined. The default JASPI factory implementation has been set in the server runtime. However, JASPI might not function for a client.

To resolve, set the fully qualified class name of the default JASPI factory implementation class as the value for the property authconfigprovider.factory in the java.security file as in below:

```
#
# The fully qualified class name of the default JASPI factory implementation class.
#
authconfigprovider.factory=com.ibm.ws.security.jaspi.ProviderRegistry
```

Error Message: SECJ0352E: Could not get the users matching the pattern {0} because of the following exception {1}

This authentication failure message displays when an external user account repository is corrupted or unavailable, and WebSphere Application Server is unable to authenticate the user name in the repository. Generally, authentication error messages are followed by additional information that indicates the nature or root cause of the problem, such as:

```
Make sure the users matching the pattern exist in the registry. Contact your service representative if the problem persists.
```

This additional information might not provide a clear user action if the user account repository is corrupted or the user loses connectivity between WebSphere Application Server and an external user account repository. The external user account repository, which is referred to as a repository in this document, might be a Lightweight Directory Access Protocol (LDAP) product.

To resolve this problem, you might need to re-install the repository and verify that it installs successfully by testing the connection.

CAUTION: Proceed with the following steps only if you have ensured that all WebSphere Application Server-related configuration settings are accurate.

Complete the following steps to resolve the issue:

1. Restart both the repository and WebSphere Application Server.
2. Test the connection to the repository. If the connection attempt still fails, it might be necessary to re-install the repository.
3. If diagnostics are provided with the repository, run them to avoid having to re-install the repository.

   Attention: If the previous steps do not fix the problem, you might need to re-install the repository. Before proceeding, generate a complete list of all the configured users and groups; you will need to re-populate these fields after the re-installation.
4. If necessary, re-install the corrupted repository.
5. Populate the users and groups from your list into the newly installed repository.
6. Restart both the repository and WebSphere Application Server.
7. In the administrative console, navigate to **Security** > **Global security**, and select the appropriate user account repository. For example, select Standalone LDAP registry if you are using a stand-alone Lightweight Directory Access Protocol repository.
8. Click `Test connection` to ensure that WebSphere Application Server can connect to the repository.

Validation of LTPA token failed due to invalid keys or token type

If the security context deserialization of an LTPA token fails with a WSSecurityException containing this message: `Validation of LTPA token failed due to invalid keys or token type`, set the com.ibm.websphere.security.recoverContextWithNewKeys property to `true`.

Generate keys error when using the Profile Management tool to create a new profile

When you create a new profile using either the Profile Management tool or the command-line manageprofiles utility, an error message displays that indicates either partial success or failure. The error message, which is located in the install_dir/logs/manageprofiles/*profile_name*_create.log file, might point to an error in either the generateKeysforSingleProfile task or the generateKeysForCellProfile task.

The Profile Creation tool and the manageprofiles utility invoke several tasks. The generateKeysForSingleProfile task is invoked when you create a stand-alone application server or a deployment manager profile. The generateKeysForCellProfile task is invoked when you create a cell profile. Both of these tasks are the first tasks to invoke the wsadmin commands. Although the log indicates an error in one of these tasks, the error might actually result from a wsadmin command failure and not an error in the security tasks.

To determine the actual cause of the problem, review the information that is provided in the following log files:

- install_dir/logs/manageprofiles/*profile_name*_create.log file indicates the error code of the failure
- install_dir/logs/manageprofiles/*profile_name*/keyGeneration.log file
- install_dir/logs/manageprofiles/*profile_name*/wsadminListener.log file

Some security roles are not immediately available for a secured application where LDAP has Tivoli Access Manager enabled

In some instances, some security roles might not be immediately available when you deploy a secured application where LDAP has Tivoli Access Manager enabled.

You might see an error such as the following:

```
"Exception: java.lang.OutOfMemoryError"
```

You might be able to address this issue by doing the following:

1. Allocate more memory to the minimum and maximum java heap size.
   a. In the administrative console, select **Servers** > **server types** > **WebSphere Application servers** > **server1**.
   b. Select **Server Infrastructure** > **Java and Process Management** > **Process Definition**.
   c. Select Java Virtual Machine.
   d. Set the initial heap size to 512 MB and the maximum heap size to 1024 MB.
   e. Select `OK` and then `Save`.
   f. Restart WebSphere Application Server.
2. While WebSphere Application Sever is stopped, add some performance tuning properties for embedded Tivoli Access Manager.
   a. In the `config/cells/CELLNAME` directory, edit the `amwas.amjacc.template.properties` file by adding the following properties to it:

```
com.tivoli.pd.as.jacc.DBRefresh=0
com.tivoli.pd.as.jacc.AuthTableRemoteMode=yes
com.tivoli.pd.as.rbpf.NoUncheckedRoles=true
```

      This helps when embedded Tivoli Access Manager is re-configured
   b. Because embedded Tivoli Access Manager is already configured, you can update the generated configuration files with the above properties. For each WebSphere Application Server instance in the ND (dmgr, NAs and servers), go to the `profiles/NAME/etc/tam` directory and do the following.

      For each file that ends in amjacc.properties, add the 3 properties above:

```
com.tivoli.pd.as.jacc.DBRefresh=0
com.tivoli.pd.as.jacc.AuthTableRemoteMode=yes
com.tivoli.pd.as.rbpf.NoUncheckedRoles=true
```

For each file that ends in pdperm.properties, update the appsvr-dbrefresh property to be:

```
appsvr-dbrefresh=0
```

For each file that ends in authztable.pdperm.properties, update the appsvr-mode property to be:

```
appsvr-mode=remote
```

3. Restart the cell.

After setting domain realm to not trusted, global security settings cannot be used

If you add a trusted domain realm and later on decide to set this realm to “Not Trusted” from the administrative console, an empty inboundTrustedAuthenticationRealm entry might be generated in the `domain-security.xml` file. This empty inbound or outbound trusted realm definition in the `domain-security.xml` file blocks this domain from using global security settings.

To resolve this issue, do the following:

1. Remove the current domain.
2. Create a new domain.
3. Do not add the incorrect realm as “Trusted”.

Updated global security realm names are duplicated

When the global security realm names are updated, the realm names of the application security domain are also updated with the same realm names

In WebSphere Application Server Version 8.0, you can configure a unique instance of a federated repository at the domain level in a multiple security domain environment in addition to having an instance at the global level. However, if the federated repositories user registry is configured at the global level, or if the realm names are changed at the global level after configuring security domains, the realm names for all security domains using federated repositories are also updated. This causes all of the domains using federated repository to use the federated repository that is defined at the global level.

To resolve this issue, update security domains using federated repository with the original realm name after you create federated repositories or change realm names at the global level. The problem can be avoided if a federated repository at the global level is configured before you configure a federated repository in a security domain.

Note: When the global security realm names are updated, the realm names of the application security domain are not updated with the same realm names in Fix Pack 2.

Errors may occur when the session security feature is turned on

When the session security feature is turned on (which is the default in WebSphere Application Server Version 8.0), and multiple sessions are using the same user ID, when a user logs out of one session, another session might receive the following error when a different user who has logged in with the same user ID logs out:

```
SESN0008E: A user authenticated as anonymous has attempted to access a session owned by user:{<user>}
```

To resolve this issue, ensure that the previous user is logged out before another user logs in using the same user ID.

Note: This issue might also occur in some instances when the session security feature is not turned on. If so, the resolution is the same: ensure that the previous user is logged out before another user logs in using the same user ID.

Access problems after enabling security

Use this information if you are experiencing access problems after enabling security.

Note: This topic references one or more of the application server log files. As a recommended alternative, you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log` , `SystemErr.log`, `trace.log`, and `activity.log` files on distributed and IBM i systems. You can also use HPEL in conjunction with your native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

What kind of error are you seeing?
- "I cannot access all or part of the administrative console or use the wsadmin tool after enabling security"
- "Cannot access a web page after enabling security" on page 1015
- "Authentication error accessing a Web page" on page 1006
- "Authorization error accessing a Web page" on page 1006
- "The client cannot access an enterprise bean after enabling security" on page 1015
- "Client program never gets prompted when accessing secured enterprise bean" on page 1017
- "Cannot stop an application server, node manager, or node after enabling security" on page 1017
- "The AccessControlException exception, is reported in the SystemOut.log" on page 1008
- "After enabling single sign-on, cannot logon to the administrative console" on page 1018
- "Access problems after enabling security"
- "Name NotFoundException error occurs when initially connecting to the federated repositories" on page 1018

For general tips on diagnosing and resolving security-related problems, see the topic "Security components troubleshooting tips" on page 989.

If you do not see a problem that resembles yours, or if the information provided does not solve your problem, see Troubleshooting help from IBM.

I cannot access all or part of the administrative console or use the wsadmin tool after enabling security

- If you cannot access the administrative console, or view and update certain objects, look in the SystemOut log of the application server which hosts the administrative console page for a related error message.
- You might not have authorized your ID for administrative tasks. This problem is indicated by errors such as:
  - `[8/2/02 10:36:49:722 CDT] 4365c0d9 RoleBasedAuth A CWSCJ0305A: Role based authorization check failed for security name MyServer/myUserId, accessId MyServer/S-1-5-21-882015564-4266526380-2569651501-1005 while invoking method getProcessType on resource Server and module Server.`
  - Exception message: `"CWWMN0022E: Access denied for the getProcessType operation on Server MBean"`
  - `When running the command: wsadmin -username j2ee -password j2ee: CWWAX7246E: Cannot establish "SOAP" connection to host "BIRKT20" because of an authentication failure. Ensure that user and password are correct on the command line or in a properties file.`

  To grant an ID administrative authority, from the administrative console, click **System Administration** > **Console Users** and validate that the ID is a member. If the ID is not a member, add the ID with at least monitor access privileges, for read-only access.
- Verify that the trusted application functionality is enabled. The trusted application functionality is enabled if WebSphere Application Server has SAF access of READ to the RACF class of FACILITY, and profile of BBO.TRUSTEDAPPS.<cell short name>.<cluster short name>.

Cannot access a web page after enabling security

When secured resources are not accessible, probable causes include:

- Authentication errors - WebSphere Application Server security cannot identify the ID of the person or process. Symptoms of authentication errors include:

  On a Netscape browser:
  - `Authorization failed. Retry?` message is displayed after an attempt to log in.
  - Accepts any number of attempts to retry login and displays `Error 401` message when Cancel is clicked to stop retry.
  - A typical browser message displays: `Error 401: Basic realm='Default Realm'.`

  On an Internet Explorer browser:
  - Login prompt displays again after an attempt to log in.
  - Allows three attempts to retry login.
  - Displays `Error 401` message after three unsuccessful retries.
- Authorization errors - The security function has identified the requesting person or process as not authorized to access the secured resource. Symptoms of authorization errors include:
  - Netscape browser: “Error 403: AuthorizationFailed” message is displayed.
  - Internet Explorer:
    - “You are not authorized to view this page message” is displayed.
    - “HTTP 403 Forbidden” error is also displayed.
- SSL errors - WebSphere Application Server security uses Secure Sockets Layer (SSL) technology internally to secure and encrypt its own communication, and incorrect configuration of the internal SSL settings can cause problems. Also you might have enabled SSL encryption for your own web application or enterprise bean client traffic which, if configured incorrectly, can cause problems regardless of whether WebSphere Application Server security is enabled.
  - SSL-related problems are often indicated by error messages that contain a statement such as: ERROR: Could not get the initial context or unable to look up the starting context.Exiting. followed by javax.net.ssl.SSLHandshakeException

The client cannot access an enterprise bean after enabling security

If the client access to an enterprise bean fails after security is enabled:

- Review the steps for securing and granting access to resources.
- Browse the server JVM logs for errors relating to enterprise bean access and security. Look up any errors in the message table.

  Errors similar to `Authorization failed for /UNAUTHENTICATED while invoking` *`resource`* `securityName:/UNAUTHENTICATED;accessId:UNAUTHENTICATED not granted any of the required roles` *`roles`* indicate that:
  - An unprotected servlet or JavaServer Pages (JSP) file accessed a protected enterprise bean. When an unprotected servlet is accessed, the user is not prompted to log in and the servlet runs as UNAUTHENTICATED. When the servlet makes a call to an enterprise bean that is protected, the servlet fails.

    To resolve this problem, secure the servlet that is accessing the protected enterprise bean. Make sure that the runAs property for the servlet is set to an ID that can access the enterprise bean.
  - An unauthenticated Java client program is accessing an enterprise bean resource that is protected. This situation can happen if the file that is read by the `sas.client.props` properties file that is used by the client program does not have the `securityEnabled` flag set to `true`.

    To resolve this problem, make sure that the `sas.client.props` file on the client side has its securityEnabled flag set to `true`.

  Errors similar to Authorization failed for valid_user while invoking resource securityName:/username;accessId:xxxxxx not granted any of the required roles indicate that a client attempted to access a secured enterprise bean resource, and the supplied user ID is not assigned the required roles for that enterprise bean.

- Check that the required roles for the enterprise bean resource are accessed. View the required roles for the enterprise bean resource in the deployment descriptor of the web resource.
- Check the authorization table and make sure that the user or the group that the user belongs to is assigned one of the required roles. You can view the authorization table for the application that contains the enterprise bean resource using the administrative console.

If `org.omg.CORBA.NO_PERMISSION` exceptions occur when programmatically logging on to access a secured enterprise bean, an authentication exception has occurred on the server. Typically the CORBA exception is triggered by an underlying com.ibm.WebSphereSecurity.AuthenticationFailedException. To determine the actual cause of the authentication exception, examine the full trace stack:

1. Begin by viewing the text following WSSecurityContext.acceptSecContext(), reason: in the exception. Typically, this text describes the failure without further analysis.
2. If this action does not describe the problem, look up the Common Object Request Broker Architecture (CORBA) minor code. The codes are listed in the article titled Troubleshooting the security components reference.

   For example, the following exception indicates a CORBA minor code of 49424300. The explanation of this error in the CORBA minor code table reads:

```
authentication failed error
```

   In this case the user ID or password supplied by the client program is probably not valid:

```
org.omg.CORBA.NO_PERMISSION: Caught WSSecurityContextException in
WSSecurityContext.acceptSecContext(), reason: Major Code[0] Minor Code[0]
Message[ Exception caught invoking authenticateBasicAuthData from SecurityServer
for user jdoe. Reason: com.ibm.WebSphereSecurity.AuthenticationFailedException]
minor code: 49424300 completed:
No at com.ibm.ISecurityLocalObjectBaseL13Impl.PrincipalAuthFailReason.map_auth_fail_to_minor_code
(PrincipalAuthFailReason.java:83)
```

A `CORBA INITIALIZE` exception with `CWWSA1477W: SECURITY CLIENT/SERVER CONFIGURATION MISMATCH` error embedded, is received by client program from the server.

This error indicates that the security configuration for the server differs from the client in some fundamental way. The full exception message lists the specific mismatches. For example, the following exception lists three errors:

```
Exception received: org.omg.CORBA.INITIALIZE:
CWWSA1477W: SECURITY CLIENT/SERVER CONFIG MISMATCH:
The client security configuration (sas.client.props or outbound settings in
administrative console) does not support the server security configuration for
the following reasons:
ERROR 1: CWWSA0607E: The client requires SSL Confidentiality but the server does not
         support it.
ERROR 2: CWWSA0610E: The server requires SSL Integrity but the client does not
         support it.
ERROR 3: CWWSA0612E: The client requires client (e.g., userid/password or token),
         but the server does not support it.
     minor code: 0
     completed: No at
com.ibm.ISecurityLocalObjectBaseL13Impl.SecurityConnectionInterceptor.getConnectionKey
(SecurityConnectionInterceptor.java:1770)
```

In general, resolving the problem requires a change to the security configuration of either the client or the server. To determine which configuration setting is involved, look at the text following the `CWWSA` error message. For more detailed explanations and instructions, look in the message reference, by selecting the Reference view of the information center navigation and expanding Messages in the navigation tree.

In these particular cases:

- In ERROR 1, the client is requiring SSL confidentiality but the server does not support SSL confidentiality. Resolve this mismatch in one of two ways. Either update the server to support SSL confidentiality or update the client so that it no longer requires it.
- In ERROR 2, the server requires SSL integrity but the client does not support SSL integrity. Resolve this mismatch in one of two ways. Either update the server to support SSL integrity or update the client so that it no longer requires it.

- In ERROR 3, the client requires client authentication through a user id and password, but the server does not support this type of client authentication. Either the client or the server needs to change the configuration. To change the client configuration, modify the `SAS.CLIENT.PROPS` file for a pure client or change the outbound configuration for the server in the Security administrative console. To change the configuration for the target server, modify the inbound configuration in the Security administrative console.

Similarly, an exception like org.omg.CORBA.INITIALIZE: JSAS0477W: SECURITY CLIENT/SERVER CONFIG MISMATCH: appearing on the server trying to service a client request indicates a security configuration mismatch between client and server. The steps for resolving the problem are the same as for the JSAS1477W exceptions previously described.

Client program never gets prompted when accessing secured enterprise bean

Even though it seems that security is enabled and an enterprise bean is secured, occasions can occur when the client runs the remote method without prompting. If the remote method is protected, an authorization failure results. Otherwise, run the method as an unauthenticated user.

Possible reasons for this problem include:
- The server with which you are communicating might not have security enabled. Check with the WebSphere Application Server administrator to ensure that the server security is enabled. Access the security settings from within the Security section of the administrative console.
- The client does not have security enabled in the `sas.client.props` file. Edit the `sas.client.props` file to ensure the property com.ibm.CORBA.securityEnabled is set to `true`.
- The client does not have a ConfigURL specified. Verify that the property com.ibm.CORBA.ConfigURL is specified on the command line of the Java client, using the -D parameter.
- The specified ConfigURL does not have a valid URL syntax, or the `sas.client.props` that is pointed to cannot be found. Verify that the com.ibm.CORBA.ConfigURL property is valid. Check the Java documentation for a description of URL formatting rules. Also, validate that the file exists at the specified path.

  Windows An example of a valid property is `C:/WebSphere/AppServer/properties/sas.client.props`.
- The client configuration does not support message layer client authentication (user ID and password). Verify that the `sas.client.props` file has one of the following properties set to `true`:
  - com.ibm.CSI.performClientAuthenticationSupported=true
  - com.ibm.CSI.performClientAuthenticationRequired=true
- The server configuration does not support message layer client authentication, which consists of a user ID and password. Check with the WebSphere Application Server administrator to verify that user ID and password authentication is specified for the inbound configuration of the server within the System Administration section of the administrative console administration tool.

Cannot stop an application server, node manager, or node after enabling security

If you use command-line utilities to stop WebSphere Application Server processes, apply additional parameters after enabling security to provide authentication and authorization information.

Use the **./stopServer -help** command to display the parameters to use.

Use the following command options after enabling security:
- `./stopServer serverName -username` *name* `-password` *password*
- `./stopNode -username` *name* `-password` *password*
- `./stopManager -username` *name* `-password` *password*

If you use the Windows service panel or the net stop command to stop the WebSphere Application Server processes and the service could not be stopped, update the existing Application Server service using additional stop arguments. You might need to end the server process from the Task Manager before updating the service. Use the `-stopArgs` and the `-encodeParams` parameters to update the service as

described in the “Updating an existing Application Server service” example in the WASService command chapter of the *Administering applications and their environment* PDF book..

After enabling single sign-on, cannot logon to the administrative console

This problem occurs when single sign-on (SSO) is enabled, and you attempt to access the administrative console using the short name of the server, for example http://myserver:*port_number*/ibm/console. The server accepts your user ID and password, but returns you to the logon page instead of the administrative console.

To correct this problem, use the fully qualified host name of the server, for example http://*myserver.mynetwork.mycompany.com*:9060/ibm/console.

Name NotFoundException error occurs when initially connecting to the federated repositories

When the server attempts an indirect lookup on the java:comp/env/ds/wimDS name and makes its initial EJB connection to the federated repositories, the following error message displays in the `SystemOut.log` file:

```
NMSV0612W: A NameNotFound Exception
```

The NameNotFoundException error is caused by the reference binding definition for the jdbc/wimDS Java Naming and Directory interface (JNDI) name in the `ibm-ejb-jar-bnd.xmi` file. You can ignore this warning message. The message does not display when the wimDS database repository is configured.

Note: For IBM extension and binding files, the .xmi or .xml file name extension is different depending on whether you are using a pre-Java EE 5 application or module or a Java EE 5 or later application or module. An IBM extension or binding file is named ibm-*-ext.xmi or ibm-*-bnd.xmi where * is the type of extension or binding file such as app, application, ejb-jar, or web. The following conditions apply:

- For an application or module that uses a Java EE version prior to version 5, the file extension must be .xmi.
- For an application or module that uses Java EE 5 or later, the file extension must be .xml. If .xmi files are included with the application or module, the product ignores the .xmi files.

However, a Java EE 5 or later module can exist within an application that includes pre-Java EE 5 files and uses the .xmi file name extension.

The `ibm-webservices-ext.xmi`, `ibm-webservices-bnd.xmi`, `ibm-webservicesclient-bnd.xmi`, `ibm-webservicesclient-ext.xmi`, and `ibm-portlet-ext.xmi` files continue to use the .xmi file extensions.

SSL errors for security

You might encounter various problems after configuring or enabling Secure Sockets Layer (SSL). You may not be able to stop the deployment manager after configuring the SSL. You may not be able to access resource using HTTPS. The client and the server may not be able to negotiate the proper level of security. The problems mentioned here are only a few of the possibilities. Solving these problems is imperative to the successful operation of WebSphere Application Server.

What type of problem are you having?

- “Accessing resources using HTTPS” on page 1019
- “javax.net.ssl.SSLHandshakeException - The client and server could not negotiate the desired level of security. Reason: handshake failure” on page 1019
- “javax.net.ssl.SSLHandshakeException: unknown certificate” on page 1020

Accessing resources using HTTPS

If you are unable to access resources using a Secure Sockets Layer (SSL) URL (beginning with `https:`), or encounter error messages that indicate SSL problems, verify that your HTTP server is configured correctly for SSL. Browse the welcome page of the HTTP server using SSL by entering the URL: **https://***host_name*.

If the page works with HTTP, but not HTTPS, the problem is with the HTTP server.
- Refer to the documentation for your HTTP server for instructions on correctly enabling SSL. If you are using the IBM HTTP Server or Apache, go to: http://www.ibm.com/software/webservers/httpservers/library.html. Click **Frequently Asked Questions**> **SSL**.
- If you use the IBM Key Management (IKeyman) tool to create certificates and keys, remember to stash the password to a file when creating the Key Database (KDB) file with the IBM Key Management Tool.
  1. Go to the directory where the KDB file is created, and see if an `.sth` file exists.
  2. If not, open the KDB file with the IBM Key Management Tool, and click **Key Database File > Stash Password**. The following message is displayed: `The password has been encrypted and saved in the file`.

If the HTTP server handles SSL-encrypted requests successfully, or is not involved (for example, traffic flows from a Java client application directly to an enterprise bean that is hosted by WebSphere Application Server, or the problem displays only after enabling WebSphere Application Server security), what kind of error are you seeing?
- `javax.net.ssl.SSLHandshakeException - The client and server could not negotiate the desired level of security. Reason: handshake failure`
- `javax.net.ssl.SSLHandshakeException - The client and server could not negotiate the desired level of security. Reason: unknown certificate`
- `javax.net.ssl.SSLHandshakeException - The client and server could not negotiate the desired level of security. Reason: bad certificate`

You get this error message `org.omg.CORBA.INTERNAL: EntryNotFoundException or NTRegistryImp E CWSCJ0070E: No privilege id configured for:` when programmatically creating a credential

For general tips on diagnosing and resolving security-related problems, see “Security components troubleshooting tips” on page 989

If you do not see a problem that resembles yours, or if the information provided does not solve your problem, see Troubleshooting help from IBM

javax.net.ssl.SSLHandshakeException - The client and server could not negotiate the desired level of security. Reason: handshake failure

If you see a Java exception stack similar to the following example:

```
[Root exception is org.omg.CORBA.TRANSIENT:  CAUGHT_EXCEPTION_WHILE_CONFIGURING_
SSL_CLIENT_SOCKET: CWWJE0080E:   javax.net.ssl.SSLHandshakeException - The client
and server could not negotiate the desired level of   security. Reason: handshake
failure:host=MYSERVER,port=1079 minor code: 4942F303 completed: No]   at
com.ibm.CORBA.transport.TransportConnectionBase.connect
(TransportConnectionBase.java:NNN)
```

Some possible causes are:
- Not having common ciphers between the client and server.
- Not specifying the correct protocol.

To correct these problems:
1. Review the SSL settings. In the administrative console, click **Security > SSL certificate and key management**. Under Configuration settings, click **Manage endpoint security configurations >** *endpoint_configuration_name*. Under Related items, click **SSL configurations >** *SSL_configuration_name*. You can also browse the file manually by viewing the *install_root*/`properties/sas.client.props` file.
2. Check the property that is specified by the `com.ibm.ssl.protocol` file to determine which protocol is specified.
3. Check the cipher types that are specified by the `com.ibm.ssl.enabledCipherSuites` interface. You might want to add more cipher types to the list. To see which cipher suites are currently enabled, click **Quality of protection settings (QoP)**, and look for the **Cipher Suites** property.
4. Correct the protocol or cipher problem by using a different client or server protocol and cipher selection. Typical protocols are SSL or SSLv3.
5. Make the cipher selection 40-bit instead of 128-bit. For Common Secure Interoperability Version 2 (CSIv2), set both of the following properties to false in the `sas.client.props` file, or set security level=medium in the administrative console settings:
   - com.ibm.CSI.performMessageConfidentialityRequired=false
   - com.ibm.CSI.performMessageConfidentialitySupported=false

javax.net.ssl.SSLHandshakeException: unknown certificate

If you see a Java exception stack similar to the following example, it might be caused by not having the personal certificate for the server in the client truststore file:

```
ERROR: Could not get the initial context or unable to look up the starting context.
Exiting.  Exception received: javax.naming.ServiceUnavailableException: A
communication failure occurred while attempting to obtain an initial context using
the provider url: "corbaloc:iiop:localhost:2809". Make sure that the host and port
information is correct and that the server identified by the provider url is a
running name server. If no port number is specified, the default port number 2809
is used. Other possible causes include the network environment or workstation
network configuration. [Root exception is org.omg.CORBA.TRANSIENT:
CAUGHT_EXCEPTION_WHILE_CONFIGURING_SSL_CLIENT_SOCKET: CWWJE0080E:
javax.net.ssl.SSLHandshakeException - The client and server could not
negotiate the desired level of security. Reason: unknown
certificate:host=MYSERVER,port=1940 minor code: 4942F303 completed: No]
```

To correct this problem:
1. Check the client truststore file to determine if the signer certificate from the server personal certificate is there. For a self-signed server personal certificate, the signer certificate is the public key of the personal certificate. For a certificate authority (CA)-signed server personal certificate, the signer certificate is the root CA certificate of the CA that signed the personal certificate.
2. Add the server signer certificate to the client truststore file.

javax.net.ssl.SSLHandshakeException: bad certificate

A Java exception stack error might display if the following situations occur:
- A personal certificate exists in the client keystore that is used for SSL mutual authentication.
- The signer certificate is not extracted into the server truststore file, and thus the server cannot trust the certificate whenever the SSL handshake is made.

The following message is an example of the Java exception stack error:

```
ERROR: Could not get the initial context or unable to look
up the starting context. Exiting.
Exception received: javax.naming.ServiceUnavailableException:
A communication failure occurred while attempting to obtain an
initial context using the provider url: "corbaloc:iiop:localhost:2809".
Make sure that the host and port information is correct and that the
server identified by the provider url is a running name
```

```
server. If no port number is specified, the default port number 2809
is used. Other possible causes include the network environment or
workstation network configuration.
[Root exception is org.omg.CORBA.TRANSIENT: CAUGHT_EXCEPTION_WHILE_CONFIGURING_SSL_
CLIENT_SOCKET: CWWJE0080E: javax.net.ssl.SSLHandshakeException - The client and
server could not negotiate the desired level of security. Reason:
bad certificate: host=MYSERVER,port=1940 minor code: 4942F303 completed: No]
```

To verify this problem, check the server truststore file to determine if the signer certificate from the client personal certificate is there. For a self-signed client personal certificate, the signer certificate is the public key of the personal certificate. For a certificate authority-signed client personal certificate, the signer certificate is the root CA certificate of the CA that signed the personal certificate.

To correct this problem, add the client signer certificate to the server truststore file.

org.omg.CORBA.INTERNAL: EntryNotFoundException or NTRegistryImp E CWSCJ0070E: No privilege id configured for: error when programmatically creating a credential

If you encounter the following exception in a client application attempting to request a credential from a WebSphere Application Server using SSL mutual authentication:

```
ERROR: Could not get the initial context or unable to look up the starting context.
Exiting. Exception received: org.omg.CORBA.INTERNAL: Trace from server: 1198777258
at host MYHOST on port 0 >>org.omg.CORBA.INTERNAL: EntryNotFoundException minor
code: 494210B0 completed:
No at com.ibm.ISecurityLocalObjectBaseL13Impl.PrincipalAuthFailReason.
map_auth_fail_to_minor_code(PrincipalAuthFailReason.java:99)
```

or a simultaneous error from the WebSphere Application Server that resembles:

```
[7/31/02 15:38:48:452 CDT] 27318f5 NTRegistryImp E CWSCJ0070E: No privilege id
configured for: testuser
```

The cause might be that the user ID sent by the client to the server is not in the user registry for that server.

To confirm this problem, check that an entry exists for the personal certificate that is sent to the server. Depending on the user registry mechanism, look at the native operating system user ID or Lightweight Directory Access Protocol (LDAP) server entries.

To correct this problem, add the user ID to the user registry entry (for example, operating system, LDAP directory, or other custom registry) for the personal certificate identity.

"Catalog" tablet is blank (no item displayed) in GUI application client

This error message occurs when you install an ActiveX client sample application that uses the PlantsByWebSphere Active X to EJB Bridge.

The cause is that the server certificate is not in the client trustore that is specified in the client.ssl.props file. Although the "com.ibm.ssl.enableSignerExchangePrompt" signer property might be set to `true`, the auto-exchange prompt only supports a command-line prompt. If the sample application relies on a graphical user interface and does not provide access to a command prompt, for example using standard in and standard out, the auto-exchange prompt does not function.

Note: The applet client under the Client Technology Samples does not have access to the command prompt and it cannot see the auto-exchange prompt. Thus, the applet client cannot rely on the auto-exchange prompt feature.

To correct this problem, retrieve the certificate manually using the retrieveSigners utility.

Modifying SSL Configurations after migration using -scriptCompatibility true

After migrating using scriptCompatibility true, all attributes of the SSL configurations cannot be edited through the administrative console. In particular, the hardware cryptography settings cannot be displayed or edited.

By using the scriptCompatibility true flag, the SSL configurations are not migrated to the new format for support in the Version 6.1 and later releases. New capabilities were added that are not supported when the configurations are not migrated to the latest format. If you are migrating from a release prior to Version 6.1, you can use the convertSSLConfig task to convert your SSL configuration information to the centralized SSL configuration format.

Stand-Alone configuration fails when digital certificates are defined with the NOTRUST option

If your digital certificates are defined with the NOTRUST option, it is possible that you might receive the following error message:

```
Trace: 2008/06/18 16:57:57.798 01 t=8C50B8 c=UNK key=S2 (0000000A)
Description: Log Boss/390 Error
from filename: ./bbgcfcom.cpp
at line: 376
error message: BBOO0042E Function AsynchIOaccept failed with RV=-1, RC=124, RSN=050B0146, ?EDC5124I
Too many open files. (errno2=0x0594003D)??
```

If this error appears, enter `'D OMVS,P`. If you have a NOTRUST issue a large number appears under 'OPNSOCK'.

Check your digital certificates and make sure they are not marked with the NOTRUST option. This can occur if the certificates were created with a date beyond the expiration date of the CERTAUTH that was used to create it.

Problem when configuring an LDAP repository with SSL

When configuring an LDAP repository with SSL, you must configure the LDAP repository on the node before the node is registered with the administrative agent.

If you attempt to configure the LDAP repository after registering the node with the agent, federated repositories looks for the SSL certificates in the trust store of the administrative agent instead of in the trust store of the node.

Problem when creating a chained certificate for SHA384withECDSA

If you have certificates converted to SHA384withECDSA, and are trying to create a chained certificate from the administrative console by clicking **SSL Certificate and Key management->Key stores and certificates ->*key store* >Personal certificate**, and then create a new chained certificate, the supported key size should be 384. If it is not, the certificate cannot be created.

To resolve, enable Javascript to show the correct key size on the panel

Errors configuring SSL encrypted access for security

You might have errors returned when you are trying to configure Secure Sockets Layer (SSL) for encrypted access. This article describes some of the common errors you might encounter and makes suggestions on how to fix the problems.

What kind of error are you seeing?

- “The Java Cryptographic Extension (JCE) files were not found. error when launching iKeyman” on page 1023
- “Unable to verify MAC. error when the wrong keystore password is used” on page 1023
- ““SSL handshake failure" error when no trusted certificate is found” on page 1023

- “The certificate alias cannot be found in the keystore” on page 1024

If you do not see a problem that resembles yours, or if the information provided does not solve your problem, see Troubleshooting help from IBM for further assistance.

“The Java Cryptographic Extension (JCE) files were not found.” error when launching iKeyman

You might receive the following error when you attempt to start the iKeyman tool:

```
"The Java Cryptographic  Extension (JCE) files were not found.
Please check that the JCE files have been installed in the correct directory."
```

When you click OK, the iKeyman tool closes. To resolve this problem:

- Set the JAVA_HOME parameter so that is points to the Java Developer Kit that is shipped with WebSphere Application Server.

  AIX HP-UX Solaris For example, the command is similar to: **export JAVA_HOME=/opt/WebSphere/AppServer/java**

  Windows If WebSphere Application Server is installed on your c: drive, the command would be: set **JAVA_HOME=c:\WebSphere\AppServer\java**
- Rename the file install_dir/java/jre/lib/ext/gskikm.jar to gskikm.jar.org.

  The file is located in the install_dir/java/jre/lib/ext/ directory.

  By default, the file is located in the following directory: *app_server_rootedition_name*/java/ext.

“Unable to verify MAC.” error when the wrong keystore password is used

You might receive the following error when the keystore password is not being used correctly.

```
CWPKI0033E: The keystore located at "C:/WebSphere/AppServer/profiles/AppSrv01/etc/trust.p12"
failed to load due to the following error: Unable to verify MAC.
```

Change the Password field that references this keystore by using the correct password. The default password is WebAS. Never use this password in a production environment.

"SSL handshake failure" error when no trusted certificate is found

You might receive the following error when you attempt to add the signer to the local truststore:

```
CWPKI0022E: SSL HANDSHAKE FAILURE:  A signer with SubjectDN "CN=BIRKT40.austin.ibm.com,
 O=IBM, C=US" was sent from target host:port "9.65.49.131:9428".
```

The signer might need to be added to the local truststore C:/WASX_c0602.31/AppServer/profiles/Dmgr09/etc/trust.p12 that is located in the SSL configuration alias DefaultSSLSettings. The truststore is loaded from the SSL configuration file.

The extended error message from the SSL handshake exception is:

```
"No trusted certificate found."
```

This error indicates that the signer certificate from the specified target host and port has not been located in the specified truststore, the SSL settings, and the SSL configuration file. If this occurs in a client process, there are several things that you can do:

- Enable the signer exchange prompt.
- Run the retrieveSigners script. For more information about the location of the signer certificate see the retrieveSigners command topic in the *Securing applications and their environment PDF* book.
- Manually export the signers from the server and import them to the client.

If this issue occurs in a server process, then complete one of the following procedures:

- In the administrative console, find the target endpoint (host name and port) and determine the certificate that is used by the SSL configuration associated with it. Extract the SSL certificate to a file, and import it into the sending server truststore that is referenced in the error message.
- In the administrative console, find the sending server truststore. Go to signer certificates, add from Port, and connect directly to the target host and port, which are indicated in the message, to retrieve the signer directly into the truststore.
- Manually extract the signer from the target server and host keystore by using the iKeyman utility, and import the signer into the truststore of the server sending the certificate.

Note: As default, Websphere Application Server uses the key.p12 and trust.p12 files for any communication between Websphere Application Servers (for example between nodeagent and appserver or vice versa). If WebSphere Application Server is looking for a certificate in some file other than these, then it is possible that your application establishes the secure socket layer (SSL) configuration by using system properties, established with the System.setProperty() method. That is, SSL configurations are managed for each of your processes, and you have had to maintain individual settings for each SSL configuration in the topology.

Prior to WebSphere Application Server Version 6.1, the WebSphere Application Server management processes allowed the individually-managed SSL configurations which were set by system properties. Your pre-Version 6.1 system properties settings were processed successfully.

With WebSphere Application Server Version 6.1, central management of Secure Sockets Layer (SSL) configuration occurs. Applications that use SSL connections based on values set for system properties instead of using the centrally managed default dynamic SSL configuration can experience handshake failures. Nodeagent to appserver communications is being governed by the default dynamic SSL configuration in WebSphere Application Server Version 6.1 and not through the system properties you set. You may need to adjust your application to use the centrally managed SSL configuration of WebSphere Application Server Version 6.1.

The certificate alias cannot be found in the keystore

You might receive the following error when the certificate alias is not found in the referenced keystore:

```
CWPKI0023E: The certificate alias "default" specified by the property
com.ibm.ssl.keyStoreClientAlias is not found in KeyStore
"c:/WebSphere/AppServer/profiles/Dmgr01/config/cells/myCell/key.p12".
```

This error indicates that the certificate alias that was specified cannot be found in the referenced keystore. Either change the certificate alias or make sure that alias exists in the specified keystore.

Single sign-on configuration troubleshooting tips for security

Several common problems can occur when you configure single sign-on (SSO) between a WebSphere Application Server and a Domino server. Some such problems are: Failure to save the Domino Web SSO configuration, authentication failures when accessing a protected resource, and SSO failures when accessing a protected resource. You can take some actions to correct these error situations and restore the SSO.

- Failure to save the Domino Web SSO configuration document

  The client must find Domino server documents for the participating SSO Domino servers. The Web SSO configuration document is encrypted for the servers that you specify. The home server that is indicated by the client location record must point to a server in the Domino domain where the participating servers reside. This pointer ensures that lookups can find the public keys of the servers.

  If you receive a message stating that one or more of the participating Domino servers cannot be found, then those servers cannot decrypt the Web SSO configuration document or perform SSO.

When the Web SSO configuration document is saved, the status bar indicates how many public keys are used to encrypt the document by finding the listed servers, authors, and administrators in the document.

- Failure of the Domino server console to load the Web SSO configuration document at Domino HTTP server startup

  During configuration of SSO, the server document is configured for Multi-Server in the Session Authentication field. The Domino HTTP server tries to find and load a Web SSO configuration document during startup. The Domino server console reports the following information if a valid document is found and decrypted: HTTP: Successfully loaded Web SSO Configuration.

  If a server cannot load the Web SSO configuration document, SSO does not work. In this case, a server reports the following message: HTTP: Error Loading Web SSO configuration. Reverting to single-server session authentication.

  Verify that only one Web SSO configuration document is in the web configurations view of the Domino directory and in the $WebSSOConfigs hidden view. You cannot create more than one document, but you can insert additional documents during replication.

  If you can verify only one Web SSO configuration document, consider another condition. When the public key of the server document does not match the public key in the ID file, this same error message can display. In this case, attempts to decrypt the Web SSO configuration document fail and the error message is generated.

  This situation can occur when the ID file is created multiple times, but the Server document is not updated correctly. Usually, an error message is displayed on the Domino server console stating that the public key does not match the server ID. If this situation occurs, SSO does not work because the document is encrypted with a public key for which the server does not possess the corresponding private key.

  To correct a key-mismatch problem:
  1. Copy the public key from the server ID file and paste it into the Server document.
  2. Create the Web SSO configuration document again.

- Authentication fails when accessing a protected resource.

  If a web user is repeatedly prompted for a user ID and password, SSO is not working because either the Domino or the WebSphere Application Server security server cannot authenticate the user with the Lightweight Directory Access Protocol (LDAP) server. Check the following possibilities:
  - Verify that the LDAP server is accessible from the Domino server machine. Use the TCP/IP ping utility to check TCP/IP connectivity and to verify that the host machine is running.
  - Verify that the LDAP user is defined in the LDAP directory. Use the `idsldapsearch` utility to confirm that the user ID exists and that the password is correct. For example, you can run the following command, entered as a single line:

    You can use the OS/400® Qshell, a UNIX shell, or a Windows DOS prompt

```
% ldapsearch -D "cn=John Doe, ou=Rochester, o=IBM, c=US" -w mypassword
-h myhost.mycompany.com -p 389 -b "ou=Rochester, o=IBM, c=US" (objectclass=*)
```

    The percent character (%) indicates the prompt and is not part of the command. A list of directory entries is expected. Possible error conditions and causes are contained in the following list:
    - No such object: This error indicates that the directory entry referenced by either the user's distinguished name (DN) value, which is specified after the `-D` option, or the base DN value, which is specified after the `-b` option, does not exist.
    - Credentials that are not valid: This error indicates that the password is not valid.
    - Cannot contact the LDAP server: This error indicates that the host name or the port specified for the server is not valid or that the LDAP server is not running.
    - An empty list means that the base directory that is specified by the `-b` option does not contain any directory entries.
  - If you are using the user's short name or user ID instead of the distinguished name, verify that the directory entry is configured with the short name. For a Domino directory, verify the Short name/UserID field of the Person document. For other LDAP directories, verify the userid property of the directory entry.

– If Domino authentication fails when using an LDAP directory other than a Domino directory, verify the configuration settings of the LDAP server in the Directory assistance document in the Directory assistance database. Also verify that the Server document refers to the correct Directory assistance document. The following LDAP values that are specified in the Directory Assistance document must match the values specified for the user registry in the WebSphere Application Server administrative domain:
  - Domain name
  - LDAP host name
  - LDAP port
  - Base DN

  Additionally, the rules that are defined in the Directory assistance document must refer to the base distinguished name (DN) of the directory that contains the directory entries of the users.

  You can trace Domino server requests to the LDAP server by adding the following line to the server `notes.ini` file:

  ```
  webauth_verbose_trace=1
  ```

  After restarting the Domino server, trace messages are displayed in the Domino server console as Web users attempt to authenticate to the Domino server.

- Authorization failure when accessing a protected resource.

  After authenticating successfully, if an authorization error message is displayed, security is not configured correctly. Check the following possibilities:
  – For Domino databases, verify that the user is defined in the access-control settings for the database. Refer to the Domino administrative documentation for the correct way to specify the user's DN. For example, for the DN cn=John Doe, ou=Rochester, o=IBM, c=US, the value on the access-control list must be set as John Doe/Rochester/IBM/US.
  – For resources that are protected by WebSphere Application Server, verify that the security permissions are set correctly.
    - If granting permissions to selected groups, make sure that the user attempting to access the resource is a member of the group. For example, you can verify the members of the groups by using the following website to display the directory contents: Ldap://myhost.mycompany.com:389/ou=Rochester, o=IBM, c=US??sub
    - If you changed the LDAP configuration information (host, port, and base DN) in a WebSphere Application Server administrative domain since the permissions were set, the existing permissions are probably not valid and need to be recreated.

- SSO failure when accessing protected resources.

  If a web user is prompted to authenticate with each resource, SSO is not configured correctly. Check the following possibilities:
  1. Configure both WebSphere Application Server and the Domino server to use the same LDAP directory. The HTTP cookie that is used for SSO stores the full DN of the user, for example, cn=John Doe, ou=Rochester, o=IBM, c=US, and the domain name service (DNS) domain.
  2. Define web users by hierarchical names if the Domino directory is used. For example, update the User name field in the Person document to include names of this format as the first value: John Doe/Rochester/IBM/US.
  3. Specify the full DNS server name, not just the host name or TCP/IP address for websites issued to Domino servers and WebSphere Application Servers that are configured for SSO. For browsers to send cookies to a group of servers, the DNS domain must be included in the cookie, and the DNS domain in the cookie must match the web address. This requirement is why you cannot use cookies across TCP/IP domains.
  4. Configure both Domino and the WebSphere Application Server to use the same DNS domain. Verify that the DNS domain value is exactly the same, including capitalization. You need the name of the DNS domain in which WebSphere Application Server is configured. For additional information about configuring DNS domains for SSO, refer to the Single sign-on topic in the *Securing applications and their environment* PDF book.

5. Verify that the clustered Domino servers have the host name populated with the full DNS server name in the server document. By using the full DNS server name, Domino Internet Cluster Manager (ICM) can redirect to cluster members using SSO. If this field is not populated, by default, ICM redirects web addresses to clustered web servers by using the host name of the server only. ICM cannot send the SSO cookie because the DNS domain is not included in the web address. To correct the problem:
   a. Edit the Server document.
   b. Click **Internet Protocols** > **HTTP tab**.
   c. Enter the full DNS name of the server in the Host names field.
6. If a port value for an LDAP server is specified for a WebSphere Application Server administrative domain, edit the Domino Web SSO configuration document and insert a backslash character (\) into the value of the LDAP Realm field before the colon character (**:**). For example, replace myhost.mycompany.com:389 with myhost.mycompany.com\:389.

- Users are not logged out after the HTTP session timer expires.

  If users of WebSphere Application Server log onto an application and sit idle longer than the specified HTTP session timeout value, the user information is not invalidated and user credentials stay active until LTPA token timeout occurs.

  After you apply PK25740, complete the following steps to log out users from the application after the HTTP session has expired.

  1. In the administrative console, click **Security > Global security**.
  2. Under Custom properties, click **New**.
  3. In the Name field, enter com.ibm.ws.security.web.logoutOnHTTPSessionExpire.
  4. In the Values field, enter true.
  5. Click Apply and Save to save the changes to your configuration.
  6. Resynchronize and restart the server.

  Unexpected re-authentications: When you set the com.ibm.ws.security.web.logoutOnHTTPSessionExpire custom property to true, unexpected re-authentications might occur when you are working with multiple web applications. By default, each web application has its own unique HTTP session, but the web browser has one session cookie. To address this issue, you can change the HTTP session configuration by giving each application a unique session cookie name or path setting. As a result, each application gets its own session cookie. Alternatively, you can configure multiple web applications with the same enterprise application to share the same HTTP session. For more information, see the Assembling so that session data can be shared topic.

- Possible issues when SSO is enabled and Firefox v3.6.11 is configured to accept third-party cookies.

  If you have SSO enabled, and when using Firefox v3.6.11 one of the following is true:

  – It is configured to accept third-party cookies that are kept until they expire or until Firefox is closed
  – You have one session open but are switching to different applications
  – More than one session is opened for different applications that require different users for authorization

  you might see the following error message: `Error 403: AuthorizationFailed`.

  To resolve, clear the third-party cookies before launching a new application by doing the following:

  1. Select **Firefox Tools** > **Options** > **Privacy**.
  2. Ensure that the history is set to `Remember History`.
  3. Click on `Remove individual cookies` to delete the cookies.

  You can also close other sessions if Firefox is configured to accept third-party cookies that are kept until Firefox is closed.

Security authorization provider troubleshooting tips

This article describes the issues you might encounter using a Java Authorization Contract for Containers (JACC) authorization provider. Tivoli Access Manager is bundled with WebSphere Application Server as an authorization provider. However, you also can plug in your own authorization provider.

Tivoli Access Manager as a Java Authorization Contract for Containers authorization provider

Note: This topic references one or more of the application server log files. As a recommended alternative, you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log` , `SystemErr.log`, `trace.log`, and `activity.log` files on distributed and IBM i systems. You can also use HPEL in conjunction with your native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

You might encounter the following issues when using Tivoli Access Manager as a JACC authorization provider:

- The configuration of JACC might fail.
- The server might fail to start after configuring JACC.
- The application might not deploy properly.
- The startServer command might fail after you have configured Tivoli Access Manager or a clean uninstall did not take place after unconfiguring JACC.
- HPDIA0202w An unknown user name was presented to Access Manager error might occur.
- HPDAC0778E The specified user's account is set to invaliderror might occur.
- WASX7017E: Exception received while running file InsuranceServicesSingle.jacl error might occur.
- “Access denied exceptions accessing applications when using JACC” on page 1031

External providers for Java Authorization Contract for Containers authorization provider

You might encounter the following issues when you use an external provider for JACC authorization:

- HPDJA0506E Invalid argument: Null or zero-length user name field for the ACL entry error might occur.

The configuration of JACC might fail

If you have problems configuring JACC, check the following items:

- Ensure that the parameters are correct. For example, you do not want a number after TAM_Policy_server_hostname:7135, but you do want be a number after TAM_Authorization_server_hostname:7136 (for example, TAM_Authorization_server_hostname:7136:1).
- If a message such as “server can't be contacted” is displayed, it is possible that the host names or port numbers of the Tivoli Access Manager servers are incorrect, or that the Tivoli Access Manager servers have not started.
- Ensure that the password for the sec_master user is correct.
- Check the `SystemOut.log` file and search for the `AMAS` string to see if any error messages are present.

The server might fail to start after configuring JACC

If the server does not start after JACC is configured, check the following items:

- Ensure that WebSphere Application Server and Tivoli Access Manager use the same Lightweight Directory Access Protocol (LDAP) server.

- If the message “Policy Director Authentication failed” is displayed, ensure that:
  - WebSphere Application Server LDAP server ID is the same as the “Administrator user” in the Tivoli Access Manager JACC configuration panel.
  - Verify that the Tivoli Access Manager Administrator distinguished name (DN) is correct.
  - Verify that the password of the Tivoli Access Manager administrator has not expired and is valid.
  - Ensure that the account is valid for the Tivoli Access Manager administrator.
- If a message such as `socket can't be opened for xxxx` (where `xxxx` is a number) is displayed, take the following actions:
  1. Go to the `profile_root/etc/tam` directory.
  2. Change `xxxx` to an available port number in the `amwas.commomconfig.properties` file. If the node failed to start, change `xxx` to an available port number in the `amwas*cellName_nodeName_.properties` file. If the Application Server failed to start, change `xxxx` in the `amwas*cellname_nodeName_serverName.properties` file.

The application might not deploy properly

When you click **Save**, the policy and role information is propagated to the Tivoli Access Manager policy. This process might take some time to finish. If the save fails, you must uninstall the application and then reinstall it.

To access an application after it is installed, you must wait 30 seconds, by default, to start the application after you save.

The startServer command might fail

The startServer command might fail after you configure Tivoli Access Manager or a clean uninstall did not take place after unconfiguring JACC.

If the cleanup for JACC unconfiguration or start server fails after JACC is configured, take the following actions:

- Remove Tivoli Access Manager properties files from WebSphere Application Server.

  The following files must be removed.

```
install_root/tivoli/tam/PdPerm.properties
install_root/tivoli/tam/PdPerm.ks
profile_root/etc/tam/*
```

- Use a utility to clear the security configuration and return the system to the state it was in before you configure the JACC provider for Tivoli Access Manager. The utility removes all of the PDLoginModuleWrapper entries as well as the Tivoli Access Manager authorization table entry from the `security.xml` file, effectively removing the JACC provider for Tivoli Access Manager. Backup the `security.xml` file before running this utility.

  Enter the following commands:

```
install_root/java/jre/bin/java -classpath
"install_root/lib/AMJACCProvider.jar:CLASSPATH"
com.tivoli.pd.as.jacc.cfg.CleanSecXML fully_qualified_path/security.xml
```

“HPDIA0202w: An unknown user name was presented to Access Manager”

You might encounter the following error message if you try to use an existing user in a Local Directory Access Protocol (LDAP) user registry with Tivoli Access Manager:

```
AWXJR0008E   Failed to create a PDPrincipal for principal mgr1.:
AWXJR0007E   A Tivoli Access Manager exception was caught. Details are:
"HPDIA0202W  An unknown user name was presented to Access Manager."
```

This problem might be caused by the host name exceeding predefined limits with Tivoli Access Manager when it is configured against MS Active Directory. In WebSphere Application Server, the maximum length of the host name can not exceed 46 characters.

Check that the host name is not fully qualified. Configure the machine so that the host name does not include the host domain.

To correct this error, complete the following steps:

1. On the command line, type the following information to get a Tivoli Access Manager command prompt:

```
pdadmin -a administrator_name -p administrator_password
```

   The pdadmin administrator_name prompt is displayed. For example:

   ```
   pdadmin -a administrator1 -p passw0rd
   ```

2. At the pdadmin command prompt, import the user from the LDAP user registry to Tivoli Access Manager by typing the following information:

```
user import user_name cn=user_name,o=organization_name,c=country
```

   For example:

```
user import jstar cn=jstar,o=ibm,c=us
```

After importing the user to Tivoli Access Manager, you must use the **user modify** command to set the user account to `valid`. The following syntax shows how to use this command:

```
user modify user_name account-valid yes
```

For example:

```
user modify jstar account-valid yes
```

For information on how to import a group from LDAP to Tivoli Access Manager, see the Tivoli Access Manager documentation.

"HPDAC0778E: The specified user's account is set to invalid"

You might encounter the following error message after you import a user to Tivoli Access Manager and restart the client:

```
AWXJR0008E    Failed to create a PDPrincipal for principal mgr1.:
AWXJR0007E    A Tivoli Access Manager exception was caught.
Details are: "HPDAC0778E   The specified user's account is set to invalid."
```

To correct this error, use the **user modify** command to set the user account to valid. The following syntax shows how to use this command:

```
user modify user_name account-valid yes
```

For example:

```
user modify jstar account-valid yes
```

"HPDJA0506E: Invalid argument: Null or zero-length user name field for the ACL entry"

You might encounter an error similar to the following message when you propagate the security policy information from the application to the provider using the wsadmin **propagatePolicyToJACCProvider** command:

```
AWXJR0035E   An error occurred while  attempting to add member,
               cn=agent3,o=ibm,c=us, to role AgentRole
HPDJA0506E   Invalid argument: Null or zero-length user name field for
               the ACL entry
```

To correct this error, create or import the user, that is mapped to the security role to the Tivoli Access Manager. For more information on propagating the security policy information, see the documentation for your authorization provider.

WASX7017E: Exception received while running file “InsuranceServicesSingle.jacl”

After the JACC provider and Tivoli Access Manager are enabled, when attempting to install the application, which is configured with security roles using the wsadmin command, the following error might occur:

```
WASX7017E: Exception received while running file "InsuranceServicesSingle.jacl";
exception information: com.ibm.ws.scripting.ScriptingException: WASX7111E:
Cannot find a match for supplied option:
"[RuleManager, , , cn=mgr3,o=ibm,c=us|cn=agent3,o=ibm,c=us, cn=ManagerGro
up,o=ibm,c=us|cn=AgentGroup,o=ibm,c=us]" for task "MapRolesToUsers"
```

The $AdminApp MapRolesToUsers task option is no longer valid when Tivoli Access Manager is used as the authorization server. To correct the error, change MapRolesToUsers to TAMMapRolesToUsers.

Access denied exceptions accessing applications when using JACC

In the case of Tivoli Access Manager, you might see the following error message.

```
AWXJR0044E: The access decision for Permission, {0}, was denied because either the
PolicyConfiguration or RoleConfiguration objects did not get created successfully at
application installation time. RoleConfiguration exists = {false}, PolicyConfiguration
exists = {"false"}.
```

If the access denied exceptions are not expected for the application, check the SystemOut.log files to see if the security policy information was correctly propagated to the provider.

If the security policy information for the application is successfully propagated to the provider, the audit statements with the message key SECJ0415I appear. However, if there was a problem propagating the security policy information to the provider (for example: network problems, JACC provider is not available), the SystemOut.log files contain the error message with the message keys SECJ0396E (during install) or SECJ0398E (during modification). The installation of the application is not stopped due to a failure to propagate the security policy to the JACC provider. Also, in the case of failure, no exception or error messages appear during the save operation. When the problem causing this failure is fixed, run the propagatePolicyToJaccProvider tool to propagate the security policy information to the provider without reinstalling the application. For more information about this task, see the Propagating security policy of installed applications to a JACC provider using wsadmin scripting topic in the *Securing applications and their environment* PDF book.

SPNEGO trust association interceptor (TAI) troubleshooting tips (deprecated)

Presented here is a list of trouble shooting tips useful in diagnosing Simple and Protected GSS-API Negotiation (SPNEGO) TAI problems and exceptions.

Note:

In WebSphere Application Server Version 6.1, a trust association interceptor (TAI) that uses the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) to securely negotiate and authenticate HTTP requests for secured resources was introduced. In WebSphere Application Server 7.0, this function is now deprecated. SPNEGO web authentication has taken its place to provide dynamic reload of the SPNEGO filters and to enable fallback to the application login method.

Note: This topic references one or more of the application server log files. As a recommended alternative, you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log` , `SystemErr.log`, `trace.log`, and `activity.log` files on distributed and IBM i systems. You can also use HPEL in conjunction with your native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

The IBM Java Generic Security Service (JGSS) and IBM Simple and Protected GSS-API Negotiation (SPNEGO) providers use a Java virtual machine (JVM) custom property to control trace information. The SPNEGO TAI uses the JRas facility to allow an administrator to trace only specific classes. The following important trace specifications or JVM custom properties should be used to debug the TAI using tracing.

Table 167. SPNEGO TAI trace specifications.

This table describes the SPNEGO TAI trace specifications.

Trace	Use
com.ibm.security.jgss.debug	Set this JVM Custom Property to `all` to trace through JGSS code. Messages appear in the `trace.log` file, and SystemOut.log.
com.ibm.security.krb5.Krb5Debug	Set this JVM Custom Property to `all` to trace through the Kerberos5-specific JGSS code. Messages appear in the `trace.log` file, and SystemOut.log.
com.ibm.ws.security.spnego.*	Set this trace on using the **administrative console** > **troubleshooting** > **Logging and Tracing** > **server1** > **Change Log Detail Levels** > **com.ibm.ws.security.spnego.***. Messages appear in the `trace.log` file.

Problem: WebSphere Application Server and the Active Directory (AD) Domain Controller's time are not synchronized within 5 minutes.

Symptom

```
[2/24/06 13:12:46:093 CST] 00000060 Context  2 com.ibm.ws
 .security.spnego.Context begin GSSContext accepted
[2/24/06 13:12:46:093 CST] 00000060 Context  E com.ibm.ws
 .security.spnego.Context begin
CWSPN0011E: An invalid SPNEGO token has been encountered
 while authenticating a HttpServletRequest:
0000:  60820160 06062b06 01050502 a1820154    `..` ..+. .... ...T
0010:  30820150 a0030a01 01a10b06 092a8648    0..P .... .... .*.H
0020:  82f71201 0202a282 013a0482 01366082    .... .... .:.. .6`.
0030:  01320609 2a864886 f7120102 0203007e    .2.. *.H. .... ...~
0040:  82012130 82011da0 03020105 a1030201    ..!0 .... .... ....
0050:  1ea41118 0f323030 36303232 34313931    .... .200 6022 4191
0060:  3234365a a5050203 016b48a6 03020125    246Z .... .kH. ...%
0070:  a9161b14 57535345 432e4155 5354494e    .... WSSE C.AU STIN
0080:  2e49424d 2e434f4d aa2d302b a0030201    .IBM .COM .-0+ ....
0090:  00a12430 221b0448 5454501b 1a773230    ..$0 "..H TTP. .w20
00a0:  30337365 63646576 2e617573 74696e2e    03se cdev .aus tin.
00b0:  69626d2e 636f6dab 81aa1b81 a76f7267    ibm. com. .... .org
00c0:  2e696574 662e6a67 73732e47 53534578    .iet f.jg ss.G SSEx
00d0:  63657074 696f6e2c 206d616a 6f722063    cept ion,  maj or c
00e0:  6f64653a 2031302c 206d696e 6f722063    ode:  10,  min or c
00f0:  6f64653a 2033370a 096d616a 6f722073    ode:  37. .maj or s
0100:  7472696e 673a2044 65666563 74697665    trin g: D efec tive
0110:  20746f6b 656e0a09 6d696e6f 72207374     tok en.. mino r st
0120:  72696e67 3a20436c 69656e74 2074696d    ring : Cl ient  tim
0130:  65204672 69646179 2c204665 62727561    e Fr iday , Fe brua
0140:  72792032 342c2032 30303620 61742031    ry 2 4, 2 006  at 1
0150:  3a31323a 34352050 4d20746f 6f20736b    :12: 45 P M to o sk
0160:  65776564                               ewed
```

User Action

The preferred way to resolve this issue is to synchronize the WebSphere Application Server system time to within 5 minutes of the AD server's time. A best practice is to use a time server to keep all systems synchronized. You can also add or adjust the clockskew parameter in the Kerberos configuration file.

Note: The default for the clockskew parameter is 300 seconds (or 5 minutes).

Problem: No factory available to create a name for mechanism 1.3.6.1.5.5.2.

Problem

Getting an exception: No factory available to create a name for mechanism 1.3.6.1.5.5.2. There is no factory available to process the creation of a name for the specific mechanism.

Symptom

```
[4/8/05 22:51:24:542 EDT] 5003e481 SystemOut
     0 [JGSS_DBG_PROV] Provider
       IBMJGSSProvider version 1.01 does not support
   mech 1.3.6.1.5.5.2
[4/8/05 22:51:24:582 EDT] 5003e481 ServerCredent >
        com.ibm.ws.security.spnego
    .ServerCredential initialize ENTRY
SPNEGO014: Kerberos initialization Failure:
org.ietf.jgss.GSSException, major code: 2,
        minor code: 0
 major string: Unsupported mechanism
 minor string: No factory available to create name for
  mechanism 1.3.6.1.5.5.2
 at com.ibm.security.jgss.i18n.I18NException.throwGSSException
        (I18NException.java:30)
 at com.ibm.security.jgss.GSSManagerImpl.a(GSSManagerImpl.java:36)
 at com.ibm.security.jgss.GSSCredentialImpl.add(GSSCredentialImpl
 .java:217)
 at com.ibm.security.jgss.GSSCredentialImpl.<init>(GSSCredentialImpl
 .java:264)
```

User Action

Check the `java.security` file to ensure it contains the IBMSPNEGO security provider and that the provider is defined correctly. The `java.security` file should contain a line similar to:

```
security.provider.6=com.ibm.security
 .jgss.mech.spnego.IBMSPNEGO
```

Problem: Getting an exception as the JGSS library is trying to process the SPNEGO token.

Symptom

The following error is displayed as the JGSS library is trying to process the SPNEGO token.

```
Error authenticating request. Reporting to client
Major code = 11, Minor code = 31
org.ietf.jgss.GSSException, major code: 11,
 minor code: 31
 major string: General failure, unspecified
  at GSSAPI level
 minor string: Kerberos error while decoding
  and verifying token:
    com.ibm.security.krb5.internal.KrbException,
   status code: 31
 message: Integrity check on decrypted field
 failed
```

Description

This exception is the result of encoding the ticket using one key and attempting to decode it using a different key. There are number of possible reasons for this condition:

1. The Kerberos keytab file has not been copied to the server machine once it has been regenerated.
2. The Kerberos configuration points to the wrong Kerberos keytab file.
3. The Kerberos service principal name (SPN) has been defined to the Active Directory more than once. You have another userid defined with the same SPN or defined with the same SPN with a port defined also. The following example demonstrates how this condition can occur:

SAME SPN but different user ids

```
setspn -a HTTP/myHost.austin.ibm.com user1
 setspn -a HTTP/myHost.austin.ibm.com user2
```

SAME SPN and same user ids, one without a port number, one with a port number

```
setspn -a HTTP/myHost.austin.ibm.com user
 setspn -a HTTP/myHost.austin.ibm.com:9080 user
```

User Action

If the problem is with the Kerberos keytab file, then regenerate the keytab file. If the problem is with multiple SPN definitions, then remove the extra or conflicting SPN, confirm that the SPN is no longer registered with the Active Directory, and then add the SPN. The Active Directory may need to be searched for other entries with SPNs defined that clash with the SPN.

To confirm that the SPN is not registered, the command:

```
setspn –l userid
```

should return with the following response:

```
Cannot find account userid
```

Problem: Single sign-on is not occurring.

Symptom

When tracing is enabled, the following message appears:

```
[2/27/06 14:28:04:191 CST] 00000059
SpnegoHandler <
  com.ibm.ws.security.spnego
  .SpnegoHandler handleRequest:
  Received a non-SPNEGO
  Authorization Header RETURN
```

Description

The client is returning an NT LAN manager (NTLM) response to the authorize challenge, not a SPNEGO token. This condition can be occur due to any of the following reasons:

- The client has not been configured properly.
- The client is not using a supported browser. For example, when using Microsoft Internet Explorer 5.5, SP1 responds with a non-SPNEGO authentication header.
- The user has not logged into the Active Directory domain, or into a trusted domain, or the client used does not support integrated authentication with Windows – in this case, the SPNEGO TAI is working properly.
- The user is accessing a service defined on the same machine upon which the client is running (local host). Microsoft Internet Explorer resolves the host name of the URL to `http://localhostsomeURL` instead of a fully qualified name.
- The SPN is not found in the Active Directory. The SPN must be of the format `HTTP/server.realm.com`. The command to add the SPN is

  ```
  setspn –a HTTP/server.realm.com userid
  ```

- The Kerberos service principal name (SPN) has been defined to the Active Directory more than once. You have either another user ID defined with the same SPN or another userid defined with the same SPN with a port number defined. The following categories describe these conditions:

  Same SPN but with differing user IDs

  - `setspn -a HTTP/myappserver.austin.ibm.com user1`
  - `setspn -a HTTP/myappserver.austin.ibm.com user2`

  Same SPN and same user IDs, one with a port number defined

  - `setspn -a HTTP/myappserver.austin.ibm.com user3`
  - `setspn -a HTTP/myappserver.austin.ibm.com:9080 user3`

User Action

If the SPN is defined incorrectly as `HTTP/server.realm.com@REALM.COM` with the addition of `@REALM.COM`, then delete the user, redefine the user, and redefine the SPN.

If the problem is with the Kerberos keytab file, then regenerate the keytab file.

If the problem is with either category of multiple SPN definitions, then remove the extra or conflicting SPN, confirm that the SPN is no longer registered with the Active Directory, and then add the SPN. You can search the Active Directory for other SPN entries that are causing multiple SPN definitions. The following commands are useful to determine multiple SPN definitions:

`setspn ?L userid`
Returns the message, `cannot find account userid`, if the SPN is not registered.

`setspn -L`
Displays the SPNs that exist.

Problem: Credential Delegation is not working.

Symptom

An invalid option is detected. When tracing is enabled, the following message is displayed:

```
com.ibm.security.krb5.KrbException,
 status code: 101 message:
 Invalid option in ticket request
```

Description

The Kerberos configuration file is not properly configured.

User Action

Ensure that neither renewable, nor proxiable are set to true.

Problem: Unable to get SSO working using RC4-HMAC encryption.

Symptom

Examine the following message in the trace that you receive when trace is turned on:

```
com.ibm.security.krb5.internal.crypto.
 KrbCryptoException, status code: 0
 message: Checksum error; received
 checksum does not match computed
 checksum
```

Description

RC4-HMAC encryption is not supported with a Microsoft Windows version prior to 2003 Kerberos key distribution center (KDC). To confirm this condition, examine the trace and identify where the exception is thrown. The content of the incoming ticket should be visible in the trace. Although the incoming ticket is encrypted, the SPN for the service is readable. If a Microsoft Windows version prior to 2003 KDC is used and the system is configured to use RC4-HMAC, the string representing the ticket for `userid@REALM` (instead of the expected `HTTP/hostname.realm@REALM`) is displayed. For example, this is beginning of the ticket received from a Microsoft Windows version prior to 2003 KDC:

```
0000: 01 00 6e 82 04 7f 30 82  04 7b a0 03 02 01 05 a1  ..n...0.........
0010: 03 02 01 0e a2 07 03 05  00 20 00 00 00 a3 82 03  ................
0020: a5 61 82 03 a1 30 82 03  9d a0 03 02 01 05 a1 0a  .a...0..........
0030: 1b 08 45 50 46 44 2e 4e  45 54 a2 18 30 16 a0 03  ...REALM.COM.0..
0040: 02 01 01 a1 0f 30 0d 1b  0b 65 70 66 64 77 61 73  .....0...userid
0050: 75 6e 69 74 a3 82 03 6e  30 82 03 6a a0 03 02 01  .a.f...n0..j....
```

The realm is REALM.COM. The service name is userid. A correctly formed ticket for the same SPN is:

```
0000: 01 00 6e 82 04 56 30 82  04 52 a0 03 02 01 05 a1  ..n..V0..R......
0010: 03 02 01 0e a2 07 03 05  00 20 00 00 00 a3 82 03  ................
0020: 82 61 82 03 7e 30 82 03  7a a0 03 02 01 05 a1 0a  .a...0..z.......
0030: 1b 08 45 50 46 44 2e 4e  45 54 a2 2a 30 28 a0 03  ..REALM.COM.0...
0040: 02 01 02 a1 21 30 1f 1b  04 48 54 54 50 1b 17 75  .....0...HTTP..u
0050: 73 31 30 6b 65 70 66 77  61 73 73 30 31 2e 65 70  serid.realm.com.
0060: 66 64 2e 6e 65 74 a3 82  03 39 30 82 03 35 a0 03  ...n.....90..5..
```

User Action

To correct the problem, either use the Single data encryption standard (DES) or use a Microsoft Windows 2003 Server for a KDC. Remember to regenerate the SPN, and the Kerberos keytab file.

Problem: User receives the following message when accessing a protected URL through the SPNEGO SSO.

Symptom

Examine the following message:

```
Bad Request

Your browser sent a request that this server
 could not understand.
Size of request header field exceeds server
 limit.

Authorization: Negotiate YII......
```

Description

This message is generated by the Apache/IBM HTTP Server. This server is indicating that the authorization header returned by the user's browser is too large. The long string that follows the word Negotiate (in the previous error message) is the SPNEGO token. This SPNEGO token is a wrapper of the Microsoft Windows Kerberos token. Microsoft Windows includes the user's PAC information in the Kerberos token. The more security groups that the user belongs to, the more PAC information is inserted in the Kerberos token, and the larger the SPNEGO becomes. IBM HTTP Server 2.0 (also Apache 2.0 and IBM HTTP Server 6.0) limit the size of any acceptable HTTP header to be 8K. In Microsoft Windows domains having many groups, and with user membership in many groups, the size of the user's SPNEGO token may exceed the 8K limit.

User Action

If possible, reduce the number of security groups the user is a member of. IBM HTTP Server 2.0.47 cumulative fix PK01070 allows for HTTP header sizes up to and beyond the Microsoft limit of 12K. WebSphere Application Server Version 6.0 users can obtain this fix in fixpack 6.0.0.2.
Note: Non-Apache based web servers may require differing solutions.

Problem: Even with JGSS tracing disabled, some KRB_DBG_KDC messages appear in the SystemOut.log.

Symptom
Examine the SystemOut.log and note the some KRB_DBG_KDC messages appear there even with JGSS tracing disabled.

Description
While most of the JGSS tracing is controlled by the com.ibm.security.jgss.debug property, a small set of messages are controlled by the com.ibm.security.krb5.Krb5Debug property. The com.ibm.security.krb5.Krb5Debug property has a default value to put some messages to the SystemOut.log

User Action
.To remove all KRB_DBG_KDC messages from the SystemOut.log, set the JVM property as follows:

```
-Dcom.ibm.security.krb5.Krb5Debug=none
```

Problem: An application contains a custom HTTP 401 error page, the SPNEGO TAI-generated HTTP response page is not displayed in a browser.

Symptom
When an application contains a custom HTTP 401 error page, the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) trust association interceptor (TAI)-generated HTTP response page is not displayed in a browser.

Description
When an application contains a custom HTTP 401 error page, the SPNEGO TAI-generated HTTP response page is not displayed in a browser. The custom HTTP 401 error page is displayed instead.

User Action
You can customize your HTTP 401 page to include information concerning how to configure your browser to use SPNEGO. For more information, see “Configuring the client browser to use SPNEGO TAI (deprecated)” on page 407 and “SPNEGO TAI custom properties configuration (deprecated)” on page 403.

Problem: HTTP Post parameters are lost during interaction with the SPNEGO TAI, when stepping down to userid/password login.

Symptom

Note that HTTP Post parameters are lost during interaction with the SPNEGO TAI, when stepping down to userid/password login.

"Stepping down to userid/password login" means that the Microsoft Internet Explorer tries to respond initially with a SPNEGO token. If this response is unsuccessful, then the Microsoft Internet Explorer tries to respond with a NTLM token that is obtained through a userid/password challenge.

Description

The Microsoft Internet Explorer maintains state during a user's request. If a request was given the response of an "HTTP 401 Authenticate Negotiate", and the browser responds with a NTLM token obtained through a userid/password challenge, the browser resubmits the request. If this second request is given a response of an HTML page containing a redirection to the same URL but with new arguments (via Javascript) then the browser does not resubmit the POST parameters.
Note: To avoid this problem, it is critical to NOT perform the automatic redirection. If the user clicks on a link, the problem does not occur.

User Action

The browser responds to the Authenticate/Negotiate challenge with an NTLM token, not an SPNEGO token. The SPNEGO TAI sees the NTLM, and returns back a HTTP 403 response, along with the HTML page. When the browser runs the Javascript redirTimer function, any POST of GET parameters that were present on the original request are lost.

By leveraging the SPN<id>.NTLMTokenReceivedPage property, an appropriate message page can be returned to the user. The default message that is returned (in the absence of a user defined property) is:

```
"<html><head><title>An NTLM Token was Received.
</title></head>"
 + "<body>Your browser configuration is
  correct, but you have not logged into
  a supported Windows Domain."
 + "<p>Please login to the application using
  the normal login page.</html>";
```

Using the SPN<id>.NTLMTokenReceivedPage property, you can customize the exact response. It is critical that the returned HTML not perform a redirection.

When the SPNEGO TAI has been configured to use the shipped default HTTPHeaderFilter class as the SPN<id>.filterClass, then the SPN<id>.filter can be used to allow the second request to flow directly to the normal WebSphere Application Server security mechanism. In this way, the user experiences the normal authentication mechanism.

An example of such a configuration follows showing the required SPNEGO TAI properties necessary and the HTML file content.

```
****** SPNEGO TAI Property Name ******                  ****** HTML File Content ******
com.ibm.ws.security.spnego.SPN1.hostName               server.wasteched30.torolab.ibm.com
com.ibm.ws.security.spnego.SPN1.filterClass            com.ibm.ws.security.spnego.HTTPHeader
com.ibm.ws.security.spnego.SPN1.filter                 request-url!=noSPNEGO
com.ibm.ws.security.spnego.SPN1.NTLMTokenReceivedPage  File:///C:/temp/NTLM.html
```

Note: Observe that the filter property instructs the SPNEGO TAI to NOT intercept any HTTP request that contains the string "noSPNEGO".

Here is an example of a generating a helpful response.

```
<html>
<head>
<title>NTLM Authentication Received </title>
<script language="javascript">
 var purl=""+document.location;
if (purl.indexOf("noSPNEGO")<0) {
  if(purl.indexOf('?')>=0) purl+="&noSPNEGO";
  else purl+="?noSPNEGO";
 }
</script>
</head>
<body>
<p>An NTLM token was retrieved in response to
 the SPNEGO challenge. It is likely that
you are not logged into a Windows domain.<br>
Click on the following link to get the
requested website.
<script language="javascript">
 document.write("<a href='"+purl+"'>");
 document.write("Open the same page using
  the normal authentication mechanism.");
 document.write("</a><br>");
</script>
You will not automatically be redirected.
</body>
</html>
```

Problem: The trust association interceptor (TAI) does not call the initialize(Properties) method

Tracing might show that TAI is loaded, but that the initialize(Properties) method is not called. Only the getVersion() method appears to be called during startup.

WebSphere's TAI processing only calls initialize(Properties) when there are custom properties defined for the TAI.

To fix this issue, define an unused TAI custom property, such as com.ibm.issw.spnegoTAI.NumberOfServers=0.

Problem: The trust association interceptor (TAI) is not loading properly

Tracing might show that TAI is not loading and the following exception text is received:

```
SPNEGO014: Kerberos initialization Failure: org.ietf.jgss.GSSException, major code: 13, minor code: 0
 major string: Invalid credentials
 minor string: SubjectKeyFinder: no JAAS Subject
 at com.ibm.security.jgss.i18n.I18NException.throwGSSException(I18NException.java:12)
...
```

A possible cause for this is that the JVM custom property javax.security.auth.useSubjectCredsOnly is not set to a value of `false`.

To fix this issue, define a JVM custom property on each JVM that is enabled for the TAI, javax.security.auth.useSubjectCredsOnly=false.

Problem: JACL scripts default characters for adding trust association interceptor (TAI) parameters can cause issues

JACL scripts for adding TAI parameters accept positional parameters. To accept the defaults, a "." is specified. On some WebSphere platforms, if you specify a "." it can cause the property to be added with a value of ".".

Always (regardless of platform), confirm that the properties added are as expected using the administrative console. If they are not, manually correct them.

SPNEGO troubleshooting tips

You can securely negotiate and authenticate HTTP requests for secured resources in WebSphere Application Server by using the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO). This article describes the issues you might encounter using Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) as the web authentication service for WebSphere Application Server.

SPNEGO issues and their possible solutions

Note: This topic references one or more of the application server log files. As a recommended alternative, you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log` , `SystemErr.log`, `trace.log`, and `activity.log` files on distributed and IBM i systems. You can also use HPEL in conjunction with your native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

The following are some issues you might encounter when you use SPNEGO as the web authentication service for WebSphere Application Server and their possible solutions.

- "Unable to resolve the Kerberos principal name" on page 1039
- "WebSphere Application Server and the time on the Active Directory (AD) domain controller are not synchronized within 5 minutes" on page 1039
- "No factory is available to create name for mechanism 1.3.6.1.5.5.2" on page 1040
- " A Kerberos error is received while decoding and verifying the SPNEGO token" on page 1040

Unable to resolve the Kerberos principal name

If you are unable to resolve the Kerberos principal name, as shown in the following trace example:

```
[11/11/03 1:42:29:795 EST] 1d01b21e GetKrbToken   > Negotiation (GSS): Begin handshake
[11/11/03 1:42:29:795 EST] 1d01b21e Context       > GSS Context init, servername:HTTP@johnwang5.jwcmd.com
[11/11/03 1:42:29:866 EST] 1d01b21e TraceNLS      u No message text associated with key Error.getting.the.Token,
.GSS.Exception:org.ietf.jgss.GSSException,.major.code:.13,.minor.code:.0
 major.string:.Invalid.credentials
 minor.string:.Cannot.get.credential.from.JAAS.Subject.for.principal:.HTTP/192.168.0.4@168.0.4 in bundle
com.ibm.ejs.resources.security
[11/11/03 1:42:29:866 EST] 1d01b21e GetKrbToken   E Error getting the Token, GSS Exception:org.ietf.jgss.GSSException,
major code: 13, minor code: 0
 major string: Invalid credentials
 minor string: Cannot get credential from JAAS Subject for principal: HTTP/192.168.0.4@168.0.4
[11/11/03 1:42:29:876 EST] 1d01b21e TraceNLS      u No message text associated with key SpnegoTAI.exits.due.to.an.exception.
in bundle com.ibm.ejs.resources.security
[11/11/03 1:42:29:876 EST] 1d01b21e SpnegoTAI     E SpnegoTAI exits due to an exception.
```

add the IP address of the server in its host file. You must also recycle the application server to load the new host file.

WebSphere Application Server and the time on the Active Directory (AD) domain controller are not synchronized within 5 minutes

The **trace.log** file for this issue is similar to the following:

```
[11/11/03 1:44:09:499 EST] 1d01b21e GetKrbToken   > Negotiation (GSS): Begin handshake
[11/11/03 1:44:09:499 EST] 1d01b21e Context       > GSS Context init, servername:HTTP@backendrc4.ibm.net
[11/11/03 1:44:09:499 EST] 1d01b21e Context       > GSS Context init, done.
[11/11/03 1:44:09:679 EST] 1d01b21e SpnegoTAI     > Server response token as follows...
0000:  6082014f 06062b06 01050502 a1820143     `?.O..+.....i?.C
0010:  3082013f a0030a01 01a10b06 092a8648     0?.? ....i...*?H
0020:  82f71201 0202a282 01290482 01256082     ?÷....¢?.).?.%`?
0030:  01210609 2a864886 f7120102 0203007e     .!..*?H?÷......~
0040:  82011030 82010ca0 03020105 a1030201     ?..0?.. ....i...
0050:  1ea41118 0f323030 33313131 31303634     .¤...20031111064
0060:  3430395a a5050203 0a3548a6 03020125     409Z¥....5H¦...%
0070:  a90b1b09 4a57434d 442e434f 4daa2630     ©.....IBM.NET@&0
0080:  24a00302 0100a11d 301b1b04 48545450     $ ....i.0...HTTP
0090:  1b136a6f 686e7761 6e67352e 6a77636d     ..backendrc4.ibm
00a0:  642e636f 6dab81ab 1b81a86f 72672e69     .net.«?«.?¨org.i
00b0:  6574662e 6a677373 2e475353 45786365     etf.jgss.GSSExce
```

```
00c0:  7074696f 6e2c206d 616a6f72 20636f64     ption, major cod
00d0:  653a2031 302c206d 696e6f72 20636f64     e: 10, minor cod
00e0:  653a2033 370a096d 616a6f72 20737472     e: 37..major str
00f0:  696e673a 20446566 65637469 76652074     ing: Defective t
0100:  6f6b656e 0a096d69 6e6f7220 73747269     oken..minor stri
0110:  6e673a20 436c6965 6e742074 696d6520     ng: Client time
0120:  54756573 6461792c 204e6f76 656d6265     Tuesday, Novembe
0130:  72203131 2c203230 30332061 7420313a     r 11, 2003 at 1:
0140:  33353a30 3120414d 20746f6f 20736b65     35:01 AM too ske
0150:  776564                                  wed
```

You can fix this issue in one of two ways. The preferred method is to synchronize the WebSphere system time to within 5 minutes of the time of the AD server. A best practice is to use a time server to keep all of the systems synchronized. Alternatively, you can also add or adjust the clockskew parameter in the Kerberos configuration file. Note that the default is 300 seconds (5 minutes).

No factory is available to create name for mechanism 1.3.6.1.5.5.2

If the `systemout.log` file contains an exception error similar to the following:

```
[4/8/05 22:51:24:542 EDT] 5003e481 SystemOut     0 [JGSS_DBG_PROV] Provider IBMJGSSProvider version 1.01
does not support mech 1.3.6.1.5.5.2
[4/8/05 22:51:24:582 EDT] 5003e481 ServerCredent E com.ibm.issw.spnegoTAI.ServerCredential initialize() SPNEGO014:
Kerberos initialization Failure: org.ietf.jgss.GSSException, major code: 2, minor code: 0
 major string: Unsupported mechanism
 minor string: No factory available to create name for mechanism 1.3.6.1.5.5.2
 at com.ibm.security.jgss.i18n.I18NException.throwGSSException(I18NException.java:30)
 at com.ibm.security.jgss.GSSManagerImpl.a(GSSManagerImpl.java:36)
 at com.ibm.security.jgss.GSSCredentialImpl.add(GSSCredentialImpl.java:217)
 at com.ibm.security.jgss.GSSCredentialImpl.<init>(GSSCredentialImpl.java:264)
.
.
```

Make sure that the `java.security` file contains the IBMSPNEGO security provider and is defined correctly. It should contain a line similar to the following:

```
security.provider.6=com.ibm.security.jgss.mech.spnego.IBMSPNEGO
```

A Kerberos error is received while decoding and verifying the SPNEGO token

You might receive the following exception error as the Java Generic Security Service (JGSS) library attempts to process the SPNEGO token:

```
Error authenticating request. Reporting to client
Major code = 11, Minor code = 31
org.ietf.jgss.GSSException, major code: 11, minor code: 31
 major string: General failure, unspecified at GSSAPI level
 minor string: Kerberos error while decoding and verifying token: com.ibm.security.krb5.internal.KrbException, status code: 31
 message: Integrity check on decrypted field failed
```

This error is caused when the ticket is encoded by using one key and then an attempt is made to decode the ticket by using another key. There are number of possible explanations for this:

- The keytab file has not been copied to the server machine after it has been regenerated.
- The Kerberos configuration points to the wrong keytab file.
- The SPN was defined to Active Directory more than once. This is also caused by another userid with a similarly defined SPN (either the same name or it might differ by having a port defined as part of the SPN).
- If the encryption type is DES, the password associated with the Service userid might only exist for RC4-HMAC encryption. This occurs when a new userid is created, the SPN is defined, and the keytab is generated with the `+DesOnly` option. The service ticket generated for this SPN is encrypted with one secret that does not match that found in the keytab.
- An older version of the Microsoft ktpass tool is being used. Older versions of the tool create keytab files that are incorrect and might result in this error. If you are using Windows Server 2003 as your Domain controller, use the version of **ktpass.exe** that is part of Windows Server 2003 SP 2 (specifically, version 5.2.3790.2825).

If the problem is with the keytab file, then fix it. If the problem is with multiple SPN definitions, remove the extra or conflicting SPN, confirm that the SPN is no longer registered with AD, and then add the SPN again. Read about Creating a Kerberos service principal name and keytab file for more information. The Active Directory might need to be searched for other entries with SPNs defined that clash with the SPN using an LDAP browser.

To confirm that the SPN is not registered, the following command:

```
setspn –l userid
```

should return with:

```
Cannot find account userid
```

If the userid and keytab are for DES-CBC-MD5, after you create the userid, change the password for the userid and then create the keytab file. If you are using Windows Server 2003 upgrade to the latest version of ktpass.

Single sign-on does not occur

When trace is turned on, the following error message might appear:

```
Client sent back a non-SPNEGO authentication header, SpnegoTAI exits
```

A possible reason for this error is that the client is returning an NT LAN manager (NTLM) response to the authorize challenge, not an SPNEGO token. This can occur due to one or more of the following issues:

- The client has not been properly configured.
- The client is not using a supported browser. For instance, users of Internet Explorer 5.5 SP1 respond with a non-SPNEGO authentication header.
- The user has not logged into the AD domain or into a trusted domain, or the client used does not support Integrated Authentication with Windows. In this case, the TAI is working properly.
- The user accesses a service defined on the same machine as the client is running (the localhost). Internet Explorer resolves the hostname of the URL to http://localhost<someURL> instead of to the fully-qualified name that is provided.
- The SPN is not found in the Active Directory. The SPN must be of the format HTTP/server.realm.com. The command to add the SPN is:

```
setspn –a HTTP/server.realm.com userid
```

  If the SPN is defined incorrectly as HTTP/server.realm.com@REALM.COM with the addition of @REALM.COM, then delete the user, redefine it, and then redefine the SPN.
- The hostname is resolved as a DNS Alias, not as a HOST record. Change the hostname to a HOST record.
- The account in AD that holds the ServicePrincipalName is in an AD domain that is remote from the AD domain that the user has logged into, and these domains are not Windows 2003 domains. Migrate the domains to Windows 2003, or limit SSO to users within the same domain as the ServicePrincipalName userid.

Unable to use sign-on (SSO) with RC4-HMAC encryption

When trace is turned on you might receive the following error message:

```
com.ibm.security.krb5.internal.crypto.KrbCryptoException, status code: 0
message: Checksum error; received checksum does not match computed checksum
```

Some possible reasons for this error include the following

- RC4-HMAC encryption is not supported with a Windows version prior to 2003 KDC. To confirm that this is a problem, examine the previous trace where the exception is thrown. The content of the incoming ticket should be visible in the trace. While it is encrypted, the SPN name for the service is readable. If a Windows version prior to 2003 KDC is used, and the system is configured to use RC4-HMAC, the string

representing the ticket for userid@REALMinstead of the expected HTTP/hostname.realm@REALM is shown. For example, this is beginning of the ticket received from a Windows version prior to 2003 KDC:

```
0000: 01 00 6e 82 04 7f 30 82  04 7b a0 03 02 01 05 a1  ..n...0.....
0010: 03 02 01 0e a2 07 03 05  00 20 00 00 00 a3 82 03  .............
0020: a5 61 82 03 a1 30 82 03  9d a0 03 02 01 05 a1 0a  .a...0.....
0030: 1b 08 45 50 46 44 2e 4e  45 54 a2 18 30 16 a0 03  ...REALM.COM.0..
0040: 02 01 01 a1 0f 30 0d 1b  0b 65 70 66 64 77 61 73  .....0...userid
0050: 75 6e 69 74 a3 82 03 6e  30 82 03 6a a0 03 02 01  .a.f...n0..j....
```

The realm is REALM.COM. The service name is userid. A correctly formed ticket for the same SPN is:

```
0000: 01 00 6e 82 04 56 30 82  04 52 a0 03 02 01 05 a1  ..n..V0..R......
0010: 03 02 01 0e a2 07 03 05  00 20 00 00 00 a3 82 03  .............
0020: 82 61 82 03 7e 30 82 03  7a a0 03 02 01 05 a1 0a  .a...0..z.......
0030: 1b 08 45 50 46 44 2e 4e  45 54 a2 2a 30 28 a0 03  ..REALM.COM.0...
0040: 02 01 02 a1 21 30 1f 1b  04 48 54 54 50 1b 17 75  .....0...HTTP..u
0050: 73 31 30 6b 65 70 66 77  61 73 73 30 31 2e 65 70  serid.realm.com.
0060: 66 64 2e 6e 65 74 a3 82  03 39 30 82 03 35 a0 03  ...n.....90..5..
```

To correct the problem, either use single DES encryption or use a Windows Server 2003 for a KDC. Remember to regenerate the SPN and the keytab file.

- RC-HMAC encryption does not work when the credential delegation feature is used. To determine if you have this problem, enable JGSS and Krb5 tracing. If the SPN name is correct, messages such as the following might appear:

```
[JGSS_DBG_CTX] Successfully decrypted ticket
[JGSS_DBG_CTX] Put authz info in cache
[JGSS_DBG_CTX] Session key type = rc4-hmac
...
[JGSS_DBG_CTX] Successfully decrypted authenticator
[JGSS_DBG_CTX] Error authenticating request. Reporting to client
...
Major code = 11, Minor code = 0
org.ietf.jgss.GSSException, major code: 11, minor code: 0
 major string: General failure, unspecified at GSSAPI level
 minor string: Kerberos error converting KRBCred: com.ibm.security.krb5.internal.crypto.KrbCryptoException, status code: 0
 message: Checksum error; received checksum does not match computed checksum
```

This indicates that the delegated credential contained in the SPNEGO token was not encrypted with the proper key.

Obtain APAR IY76826. This replaces **ibmjgssprovider.jar** with a version that can accept the Microsoft defined RC4 encrypted delegated credential.

- The password used when generating the keytab file with ktpass does not match the password assigned to the service account. When the password changes you should regenerate and redistribute the keys., even if it is reset to the same password.

  In addition, the ktpass tool might generate a keytab file with a non-matching password as in the following cases:

  – If the password entered to ktpass matches the password for the service account, then the produced keytab file does work.
  – If the password entered to ktpass does not match the password for the service account, and is less than 7 characters in length, ktpass stops and does not produce a keytab file.
  – If the password entered to ktpass does not match the password for the service account, and is greater than 6 characters in length, ktpass does not stop. Instead, it produces a keytab file containing an invalid key. Use of this key to decrypt a SPNEGO token produces the checksum error previously listed.

  Use a non-null password for the service account, and then use that password when invoking ktpass.
- The ktpass version 1830 (in Support Tools SP1) can produce the error in some Windows 2003 Server environments. Use the SP2 version of the tool to avoid the error.

  Use the Support Tools SP2 version of ktpass to generate the keytab file.

Credential delegation might not work due to an invalid option in the ticket request

When trace is turned on, if the following error message appears:

```
com.ibm.security.krb5.KrbException, status code: 101 message: Invalid option in ticket request
```

the Kerberos configuration file is not properly configured. Ensure that neither renewable nor proxiable are set to `true`.

Problems when accessing a protected URL through the SPNEGO single sign-on (SSO)

You might receive an error similar to the following when accessing a protected URL through the SPNEGO SSO:

```
Bad Request

Your browser sent a request that this server could not understand.
Size of request header field exceeds server limit.

Authorization: Negotiate YII......
```

This message is generated by the Apache/IBM HTTP Server, and indicates that the authorization header that your browser has returned is too large. The long string that follows the word Negotiate is the SPNEGO token. This SPNEGO token is a wrapper of the Windows Kerberos token. Windows includes the PAC information of the user in the Kerberos token. The more security groups that the user belongs to, the more PAC information is inserted in the Kerberos token, and the larger SPNEGO becomes. IBM HTTP Server 2.0 (as well as Apache 2.0 and IBM HTTP Server 6.0) limit the size of any acceptable HTTP header to be 8K. In Windows domains with many groups, and with user membership in many groups, the size of the user's SPNEGO token can exceed the 8K limit.

If possible, reduce the number of security groups that the user is a member of. IBM HTTP Server 2.0.47 cumulative fix PK01070 allows for HTTP header sizes up to and beyond the Microsoft limit of 12K.

After applying the fix you must specify the LimitRequestFieldSize parameter in the `httpd.conf` file to increase the size of allowable headers from the default of 8192.

Even with JGSS tracing disabled, some KRB_DBG_KDC messages appear in the SystemOut.log

While most of the JGSS tracing is controlled by the com.ibm.security.jgss.debug property, a small set of messages are controlled by the com.ibm.security.krb5.Krb5Debug property. The default value of the krb5 property is to emit some messages to SystemOut.log.

To remove all KRB_DBG_KDC messages from the SystemOut.log, set the JVM property to -Dcom.ibm.security.krb5.Krb5Debug=none.

ktpass is unable to find the userid

When using ktpass, you might receive an error message similar to the following:

```
DsCrackNames returned 0x2 in the name entry for server3
Failed getting target domain for specified user.
```

In an Active Directory forest, the userid lookup used by the **ktpass.exe** does not have a default domain name to be used. This does not occur when the domain controller is not in a forest.

To fix this problem, instead of specifying option `-mapUser userid`, use `-mapUser userid@domain` instead. For example, specify `–mapUser server3@WIBM.NET`.

Credential delegation does not work for any userid

If in the trace.log, an error exception similar to the following appears:

```
> com.ibm.issw.spnegoTAI.Context getDelegateCred() Entry
d com.ibm.issw.spnegoTAI.Context getDelegateCred() unable to get Delegate Credential
< com.ibm.issw.spnegoTAI.Context getDelegateCred() Exit
W com.ibm.issw.spnegoTAI.SpnegoHandler handleRequest() SPNEGO021: No delegated credentials were found for user: nauser@NA.IBM.NET
```

the domain account on which the SPN is attached does not have the “Account is trusted for Delegation” property defined.

To address this issue, ensure that the domain account does define the “Account is trusted for Delegation” property.

A user is challenged for credentials even though the browser is properly configured

A user might be challenged for credentials even though the browser is configured properly. The TAI might have obtained the user's credentials from the SPNEGO token, and the user might have failed to log in. In the trace.log an exception error similar to the following appears:

```
< com.ibm.issw.spnegoTAI.SpnegoTAI getAuthenticatedUsername(): lansche Exit
d com.ibm.issw.spnegoTAI.SpnegoTAI negotiateValidateandEstablishTrust(): Handshake finished, sending 200 :SC_OK
< com.ibm.issw.spnegoTAI.SpnegoTAI negotiateAndValidateEstablishedTrust Exit
A SECJ0222E: An unexpected exception occurred when trying to create a LoginContext. The LoginModule alias is system.WEB_INBOUND
and the exception is...
```

The userid (which is lansche in the previous example) does not exist in the registry in use by WebSphere. This problem can be caused when:

- The registry used by WebSphere is not the Active Directory domain LDAP, or Global Catalogue, but is some other virtual registry (for example, a file-based custom user registry).
- A custom IClientToServerUseridMapper implementation modifies the username such that the name it is mapped to does not exist in the registry.
- The attribute mapped to by the WebSphere LDAP User Filter property is incorrect.

To fix this problem, ensure that the user that is being asserted to WebSphere Application Server by the TAI is the configured WebSphere registry.

A user using the Novell client cannot authenticate using SPNEGO

If a user using the Novell client cannot authenticate using SPNEGO they might receive a “An NTLM token is received.” message.

The user might have logged into the Novell Client but did not perform a Windows Kerberos login (this can be confirmed using the Kerbtray utility). If a user has logged onto the Windows domain and has a Kerberos ticket, the user cannot utilize SPNEGO authentication.

To fix this problem, remove the Novell client and use the default Windows domain login.

Accessing SPNEGO sites via some caching proxy servers can cause SPNEGO authentication issues

If you access SPNEGO sites via some caching proxy servers you might not be able to authenticate using SPNEGO. The message “SPNEGO authentication not supported on this client” might be displayed.

It is possible that the caching proxy is changing the hostname that returns on the HTTP 401 Authenticate Negotiate response.

If you have this issue, contact your proxy vendor for a possible solution.

Virtual Private Networks (VPN) software and firewalls might interfere with SPNEGO operations

You might experience problems with VPN software and firewalls that might interfere with SPNEGO operations.

To resolve these issues, contact your VPN and or firewall vendors for any configuration changes that might be necessary.

Possible browser issue when accessing a SPNEGO protected application

There might be a browser issue if you log onto a domain machine using one password (for example, `passwordA`) and then log onto a second domain machine by changing your original password (for example, you might change your password on the second domain machine to `passwordB`).

Once you return to the original domain machine, you might not be able to obtain either a SPNEGO/Kerberos or an NTLM response to the Negotiate challenge. After two attempts, the browser displays an HTTP 404 error message.

To resolve this issue, log off the original domain machine and log back on with the new password (`passwordB`).

Possible browser issue with Internet Explorer 6.0

When WebSphere Application Server is configured with SPNEGO and fallback is enabled for a request, Internet Explorer 6.0 might fail to login to the form login pages.

To avoid this situation, complete one of the following actions:

- From the **Global security** > **SPNEGO Web Authentication** panel, deselect the **Allow fall back to application authentication mechanism** option if it is selected.
- Upgrade to Internet Explorer Version 7.0
- Configure Internet Explorer Version 6.0 to use a different authentication page. The issue is with the basic authentication versus the form login authentication preference.

Error pages defined for the NTLMTokenReceivedPage or the SpnegoNotSupportedPage properties do load from an http:// URL

The error pages defined for the NTLMTokenReceivedPage or the SpnegoNotSupportedPage properties do load from an http:// URL. The following trace message might appear:

```
Could not load the SPNEGO not supported content, going with the default content.
Exception received: java.net.ProtocolException: Server redirected too many  times (20)
```

This issue occurs when the loaded file performs an automatic redirect. It is not possible to both load the file from a web server and also use an automatic redirection

To resolve this issue, load the content from a file:/// URL, not an http:// URL.

A client browser single sign-on (SSO) attempt fails to authenticate

An error can occur when aclient browser single sign-on (SSO) attempt fails to authenticate with WebSphere Application Server when you use a SPNEGO token with Microsoft Internet Security Acceleration Server

When tracing is enabled, the following messages exist:

```
com.ibm.ws.security.spnego.SpnegoHandler isAuthHeaderNotSPNEGO
ENTRY Negotiate
```

```
com.ibm.ws.security.spnego.SpnegoHandler isAuthHeaderNotSPNEGO
Client sent back a non-SPNEGO authentication header
```

When a Microsoft Internet Security Acceleration Server (ISA) exists between a client browser and WebSphere Application Server, ISA might intercept the SPNEGO authentication header from the client

browser request. ISA converts the SPNEGO object identifier (OID) to a Kerberos OID. The authentication attempt with WebSphere Application Server fails because the SPNEGO OID has been converted and is now missing.

For information about how to fix this issue, see the “Users cannot access a web site that is published in ISA Server 2006 if the web site accepts only the SPNEGO authentication package” topic on the Microsoft Corporation Support site.

Microsoft Windows Version 7 and Internet Explorer Version 8 disables DES encryption type by default

If you are using Microsoft Windows Version 7 with Internet Explorer Version 8, and you cannot get SPNEGO Single Sign On (SSO) to function, it could be because Windows Version 7 disabled DES encryption type for Kerberos by default. When trace is turned on the following message appears:

```
Client sent back a non-SPNEGO authentication header....
```

It is recommended that you change your encryption type to RC4-HMAC or to AES. If you still choose to use the DES encryption type, however, you must refer to the Windows 7 documentation for help on how to enable the DES encryption type.

The following is an example of how to change your encryption type from DES to RC4:

1. Make sure the Microsoft Active Directory account that you use to map to the SPN does not have the `Use DES encryption type for this account` box checked. In the Microsoft Active Directory machine:
   a. Click **Start-** > **Programs->Administrative Tools** > **Active Directory Users and Computers** > **Users**.
   b. Click on the Microsoft Active Directory account that you use to map to the SPN.
   c. Select the account, and then make sure that the `Use DES encryption type for this account` box is not checked.
2. Reset the password for the Microsoft Active Directory account that you use to map to the SPN. You can reset it to the same password.
3. Regenerate the keytab with the RC4 encryption type.
4. Copy the new keytab file to the WebSphere Application Server servers.
5. Update the Kerberos configuration (krb5.ini/krb5.conf) files to list RC4 first for the default_tkt_enctypes and default_tgs_enctypes attributes.

   For example:

```
default_tkt_enctypes = rc4-hmac des-cbc-md5
default_tgs_enctypes = rc4-hmac des-cbc-md5
```

   .
6. Stop and restart all WebSphere Application Server servers.

Note: If you have more than one Microsoft Active Directory account that you use to map to different SPNs, then you must repeat steps 1 through 3 for each SPN and the merging of all the keytab files.

Establishing an unrestricted policy then using AES256 encryption

You can use AES256 encryption after first establishing an unrestricted policy. Follow these steps:

1. Stop the application server.
2. Download and install the new policy files.

   Important: Your country of origin might have restrictions on the import, possession, use, or re-export to another country, of encryption software. Before downloading or using the unrestricted

policy files, you must check the laws of your country, its regulations, and its policies concerning the import, possession, use, and re-export of encryption software to ensure compliance.

a. Click on the appropriate SDK level.
b. Scroll down the page then click **IBM SDK policy files**. The unrestricted JCE policy files for SDK web site displays.
c. Click **Sign in** and provide your IBM.com ID and password.
d. Select **unrestricted JCE policy files for SDK** and click **Continue**.
e. View the license and click **I Agree** to continue.
f. Extract the unlimited jurisdiction policy files that are packaged in the ZIP file. The ZIP file contains a `US_export_policy.jar` file and a `local_policy.jar` file.
g. In your WebSphere Application Server installation, go to the `$JAVA_HOME/lib/security` directory and back up your existing `US_export_policy.jar` and `local_policy.jar files`.
h. Replace your `US_export_policy.jar` and `local_policy.jar` files with the two files that you downloaded from the IBM.com web site.

Note: Take a backup before replacing these files. An example of a path that would be used is `WAS_Install/java/jre/lib/security .`

3. Start the application server.

Chapter 14. Directory conventions

References in product information to *app_server_root*, *profile_root*, and other directories imply specific default directory locations. This article describes the conventions in use for WebSphere Application Server.

Default product locations (distributed)

The following file paths are default locations. You can install the product and other components or create profiles in any directory where you have write access. Multiple installations of WebSphere Application Server products or components require multiple locations. Default values for installation actions by root and nonroot users are given. If no nonroot values are specified, then the default directory values are applicable to both root and nonroot users.

app_client_root

Table 168. Default installation root directories for the Application Client for IBM WebSphere Application Server.

This table shows the default installation root directories for the Application Client for IBM WebSphere Application Server.

User	Directory
Root	AIX `/usr/IBM/WebSphere/AppClient` (Java EE Application client only) HP-UX Linux Solaris `/opt/IBM/WebSphere/AppClient` (Java EE Application client only) Windows `C:\Program Files\IBM\WebSphere\AppClient`
Nonroot	AIX HP-UX Linux Solaris *user_home*`/IBM/WebSphere/AppClient` (Java EE Application client only) Windows `C:\IBM\WebSphere\AppClient`

app_server_root

Table 169. Default installation directories for WebSphere Application Server.

This table shows the default installation directories for WebSphere Application Server.

User	Directory
Root	AIX `/usr/IBM/WebSphere/AppServer` HP-UX Linux Solaris `/opt/IBM/WebSphere/AppServer` Windows `C:\Program Files\IBM\WebSphere\AppServer`
Nonroot	AIX HP-UX Linux Solaris *user_home*`/IBM/WebSphere/AppServer` Windows *user_home*`\IBM\WebSphere\AppServer`

component_root
: The component installation root directory is any installation root directory described in this article. Some programs are for use across multiple components—in particular, the Web Server Plug-ins, the Application Client, and the IBM HTTP Server. All of these components are part of the product package.

gskit_root
: IBM Global Security Kit (GSKit) can now be installed by any user. GSKit is installed locally inside the installing product's directory structure and is no longer installed in a global location on the target system.

Table 170. Default installation directories for GSKit.

This table shows the default installation root directory for Version 8 of the GSKit, where product_root is the root directory of the product that is installing GSKit, for example IBM HTTP Server or the web server plug-in.

User	Directory
Root and nonroot	AIX HP-UX Linux Solaris *product_root*/gsk8 Windows *product_root*\gsk8

profile_root

Table 171. Default profile directories.

This table shows the default directories for a profile named profile_name on each distributed operating system.

User	Directory
Root	AIX /usr/IBM/WebSphere/AppServer/profiles/*profile_name* HP-UX Linux Solaris /opt/IBM/WebSphere/AppServer/profiles/*profile_name* Windows C:\Program Files\IBM\WebSphere\AppServer\profiles*profile_name*
Nonroot	AIX HP-UX Linux Solaris *user_home*/IBM/WebSphere/AppServer/profiles Windows *user_home*\IBM\WebSphere\AppServer\profiles

plugins_root

Table 172. Default installation root directories for the Web Server Plug-ins.

This table shows the default installation root directories for the Web Server Plug-ins for WebSphere Application Server.

User	Directory
Root	AIX /usr/IBM/WebSphere/Plugins HP-UX Linux Solaris /opt/IBM/WebSphere/Plugins Windows C:\Program Files\IBM\WebSphere\Plugins
Nonroot	AIX HP-UX Linux Solaris *user_home*/IBM/WebSphere/Plugins Windows C:\IBM\WebSphere\Plugins

wct_root

Table 173. Default installation root directories for the WebSphere Customization Toolbox.

This table shows the default installation root directories for the WebSphere Customization Toolbox.

User	Directory
Root	AIX /usr/IBM/WebSphere/Toolbox HP-UX Linux Solaris /opt/IBM/WebSphere/Toolbox Windows C:\Program Files\IBM\WebSphere\Toolbox
Nonroot	AIX HP-UX Linux Solaris *user_home*/IBM/WebSphere/Toolbox Windows C:\IBM\WebSphere\Toolbox

web_server_root

Table 174. Default installation root directories for the IBM HTTP Server.

This table shows the default installation root directories for the IBM HTTP Server.

User	Directory
Root	AIX /usr/IBM/HTTPServer HP-UX Linux Solaris /opt/IBM/HTTPServer Windows C:\Program Files\IBM\HTTPServer
Nonroot	AIX HP-UX Linux Solaris *user_home*/IBM/HTTPServer Windows C:\IBM\HTTPServer

Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, is the user's responsibility.

APACHE INFORMATION. This information may include all or portions of information which IBM obtained under the terms and conditions of the Apache License Version 2.0, January 2004. The information may also consist of voluntary contributions made by many individuals to the Apache Software Foundation. For more information on the Apache Software Foundation, please see http://www.apache.org. You may obtain a copy of the Apache License at http://www.apache.org/licenses/LICENSE-2.0.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Intellectual Property & Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
USA

Trademarks and service marks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. For a current list of IBM trademarks, visit the IBM Copyright and trademark information Web site (www.ibm.com/legal/copytrade.shtml).

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Other company, product, or service names may be trademarks or service marks of others.

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

W

X