IBM Security Secret Server
Version 10.6

*Application Server API Guide (Deprecated)**

*** 2019: For an updated approach to access the API, see**
https://www.ibm.com/support/pages/node/872206

Contents

Last updated: November 12, 2019

Overview

Administrators have multiple options to retrieve values from Secret Server for use in applications and scripts without embedding the password. The available Application Server API options, along with standard web service integration allows scripts to access privileged credentials while keeping both the password and credentials to Secret Server secure.

TYPICAL USE CASES

- ✓ Removing embedded passwords from build scripts on build servers
- ✓ Removing embedded passwords from configuration files for applications or scripts
- ✓ Removing embedded passwords from source code in applications
- ✓ Providing accountability for the use of these passwords by scripts and applications
- ✓ Provide automated password changing by moving these passwords to the Secret Server vault

OBTAINING APPLICATION API FILES

The web services API is available to use already through your Secret Server application. To obtain files for any of the Application Server API options (listed below), please contact your account manager.

Concepts

STANDALONE JAVA API

The Standalone Java Application API is available as a Java JAR file. It exposes command line arguments that can be called. The API is deployed to a target machine with a specific application user. A script or custom application loads the Java JAR file and makes calls for specific Secrets and values without having to specify a username or password. The API is machine specific, so for each application that needs to access Secret Server, an API needs to be installed with a new Application User.

INTEGRATED JAVA API

The Standalone Java JAR file can be used as a library in a Java application. Instead of loading the Java API library and calling it through command line arguments, the Java application can make direct calls to the exposed interface.

INTEGRATED .NET CONFIGURATION API

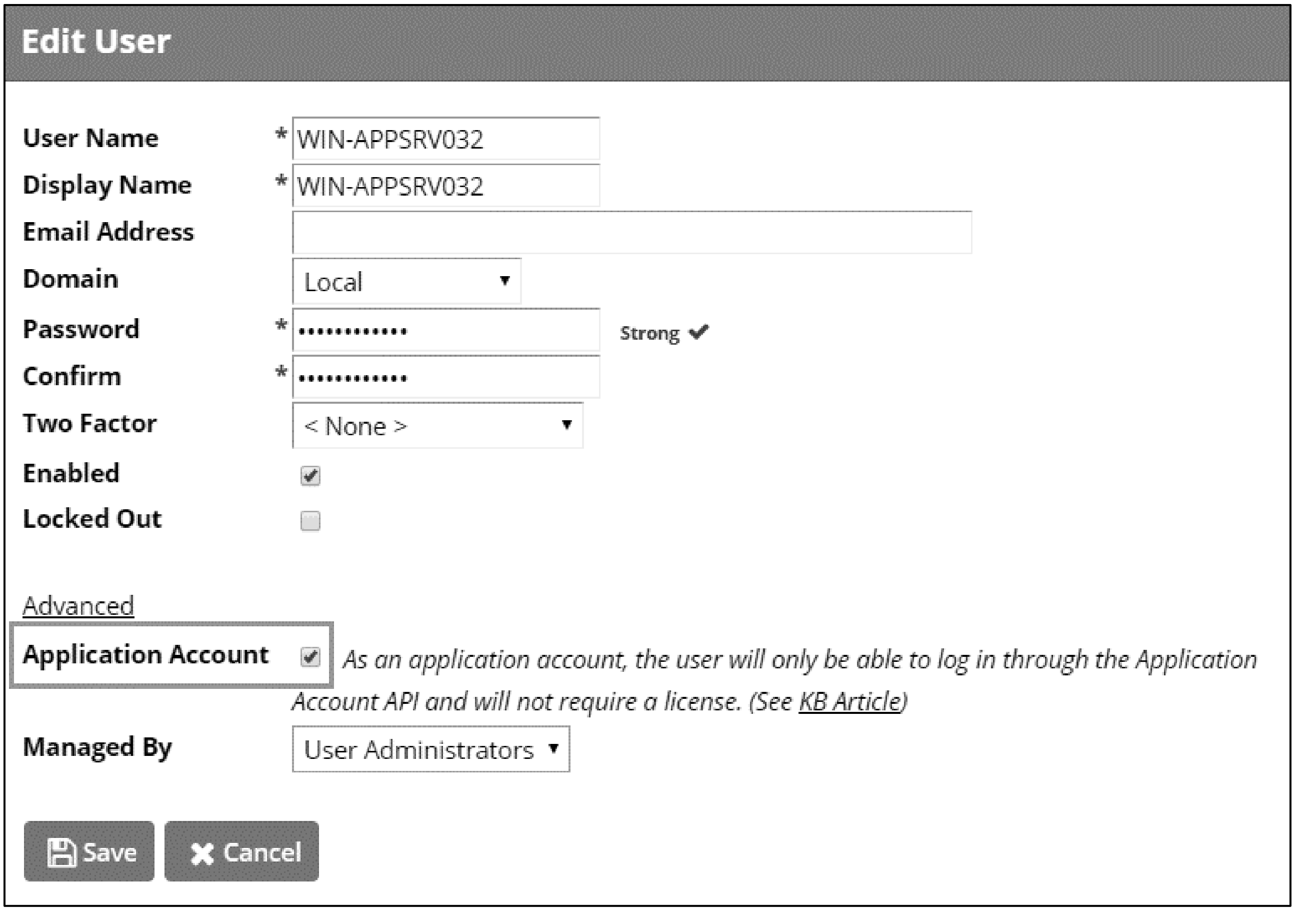
The .NET Configuration File API allows application developers and administrators to automatically replace hardcoded passwords and credentials in .NET web application configuration files with values from Secret Server with no code changes. The .NET API is installed in the bin file directory and stores credential information to authenticate to Secret Server in machine specifically encrypted files.

A developer or administrator can replace sensitive values in the web.config file using tokens. Then when the application loads the Integrated .NET API will intercept calls to retrieve those values and return the corresponding data from Secret Server. If Secret Server cannot be reached, it will return them from secure local storage. This means that no code changes are required to the applications to support the API use (simply drop in the .dll and replace the sensitive passwords with tokens).

APPLICATION USER

An Application User is a user that can only authenticate through the Application Server API or REST and SOAP web services, and is a local user to the Secret Server instance. The initial password for the Application User is set by the administrator creating it, but after it is used to install an Application API, the API resets its password to a new one based on a random seed and machine info. The use of the Application User can also be locked to a specific IP address as an optional during configuration. Application Users do not count against the user license count. If using an application account purely for SOAP or REST calls you control the account's credentials.

To create an application user in Secret Server go to **ADMIN | Users** and click **Create New.** On the User creation page expand the advanced section and check the box for **Application Account.**



Edit User

User Name	* WIN-APPSRV032
Display Name	* WIN-APPSRV032
Email Address	
Domain	Local ▾
Password	* •••••••••••• Strong ✓
Confirm	* ••••••••••••
Two Factor	< None > ▾
Enabled	☑
Locked Out	□

Advanced

Application Account	☑ *As an application account, the user will only be able to log in through the Application Account API and will not require a license. (See KB Article)*
Managed By	User Administrators ▾

WEB SERVICES

The Application Server API uses Secret Server's web services to retrieve passwords. The web services can be viewed at:

http://yoursecretserverinstallation/webservices/sswebservice.asmx

For getting the WSDL definition, append *?wsdl* to the end of the URL:

http://yoursecretserverinstallation/webservices/sswebservice.asmx?wsdl

Standalone Java API

To securely access credentials from almost any programming or scripting language, install the Standalone Java API on an application server. The user credentials are managed by the API, so no hardcoded values are necessary.

SYSTEM REQUIREMENTS

These are requirements for the client machine where the Java JAR file is executed, the Secret Server web server does not need Java installed.

- ✓ **Java Runtime:** JRE's 7 & 8 have been tested. Other versions of Java may work but are not validated.
- ✓ **Optional**: By default the JRE only allows AES-128, which JAR file uses to encrypt its data files. If you want to use AES-256 you can install with an additional command argument. This requires that specific Java JCE policy files are in place, which can be downloaded for JRE 7 here and JRE 8 here.

CONFIGURATION

1. An administrator creates an Application User in Secret Server (as a local user within the application) for a target application server.
2. The Application Server API is installed through the command line on the target server (single command is needed).
3. During the installation, the admin specifies the username and password of the Application User.
4. Using host info, the Application Server API generates a new password for the Application User, changes it to a random password in Secret Server, and stores the user info and information to recalculate the password in encrypted files on the disk.
5. Whenever a call to Secret Server is made that requires authentication, the API regenerates the password based on the machine identifier and the information in the encrypted files.

For further security, an administrator can lock down the Application User in Secret Server to a specific IP Address or IP Address range, and access to the API file can also be limited using file system permissions. The combination of these measures with the machine specific password generation means that the API cannot be simply copied to another system and used.

EXAMPLE USAGE

Install

Installation has several parameters

- ✓ **Username:** Username of the application api user account in Secret Server
- ✓ **Password:** Current password of the account
- ✓ **URL:** Secret Server URL
- ✓ **Restrict IP (Y/N):** Whether the account should only be allowed to login from this server's IP address
- ✓ **AES256 (Y/N):** Optional parameter to use AES-256 to encrypt the console's data files if the proper Java JRE policy files are rather than the default AES-128
- ✓ **AppAccount Password Rotation (-r):** Optional parameter (in hours) for whether generating a token with **-t** should periodically force a password change on the Secret Server application account. The last password change time is checked when a **-t** call is made. If the last change time exceeds the change interval the password will be changed on the app account. This is available as of version 1.1.6

  NOTE: A password change will invalidate any previously issued tokens. If multiple applications are accessing the same secretserver-jconsole.jar file to generate web service tokens then anytime the password is changed new tokens for all apps will have to be created.
- ✓ **Silent (-s):** Optional parameter for silent install with no user prompting.

Install without restricting to IP

```
C:\Program Files\java\jre7\java -jar secretserver-jconsole.jar -i appaccount apppassword https://server01/secretserver N
```

Install without restricting IP and forcing a password change every 12 hours.

```
C:\Program Files\java\jre7\java -jar secretserver-jconsole.jar -i appaccount apppassword https://server01/secretserver N
-r12
```

Silent install with IP restriction

```
C:\Program Files\java\jre7\java -jar secretserver-jconsole.jar -i appaccount apppassword https://server01/secretserver Y
-S
```

Get Field Value

Retrieve a specific value using "-s"

```
C:\Program Files\java\jre7\java -jar secretserver-jconsole.jar -s (SecretId) (FieldName)
```

Get Field Value by Secret Name

Retrieve a specific value using "-n"

```
C:\Program Files\java\jre7\java -jar secretserver-jconsole.jar -n (SecretName) (FieldName) –p(FolderName)
```

When listing the –p(FolderName), the format should be:

```
-pRootFolder\\Subfolder
```

There should be no space between –p argument and the folder name. There should be no \ before the root folder. Also, all subfolders should be separated by \\.

Get Multiple Field Values

Retrieve multiple field values using "-v"
C:\Program Files\java\jre7\java -jar secretserver-jconsole.jar -v (SecretId) (Separator) (FieldName1) (FieldName2)

Get File Attachment

Retrieve a file attachment and save it or read it as output "-f"

Download to file
C:\Program Files\java\jre7\java -jar secretserver-jconsole.jar –f (SecretId) (FieldName) –p(Save File Path)

Output as a Base64 String
C:\Program Files\java\jre7\java -jar secretserver-jconsole.jar –f (SecretId) (FieldName)

Get SOAP Token

Retrieve a user token for the application account to access SOAP web services
C:\Program Files\java\jre7\java -jar secretserver-jconsole.jar -t

Get REST Token

Retrieve a user token for the application account to access REST web services
C:\Program Files\java\jre7\java -jar secretserver-jconsole.jar -rt

Example Code in Perl

```perl
# Change secretid = 8 value to match your desired Secret Id
my $secretid = 8;
my $fieldname = 'Password'; # just the field name, not an embedded pwd
my $result = `java -jar secretserver-jconsole.jar -s $secretid $fieldname`;
my $exitCode = $?>>8; # non-zero if an error occurred
if($exitCode == 0){
      print $result;
}
else {
      print "Error: $result";
}
```

Integrated Java API

By referencing the available API Jar file, a Java application can use the same available methods as the Standalone Java API through direct calls. Note that BusinessInitializer.initialize method call prior to making any calls to Utilities. The initialize method takes the installation path of the jar file so it knows where to load its configuration files from.

```
import thycotic.secretserver.jconsole.service.* ;
import thycotic.secretserver.business.utilities.BusinessInitializer;

public class TestJavaAPI {
      /**
      * @param args
      */

      static int secretID  = 1;

      public static void main(String[] args) {
            BusinessInitializer.initialize("c:\jconsoleinstall\");

            System.out.println("user.dir := " + System.getProperty("user.dir"));

            IUtilities util = new Utilities();

        try {
            String user     = util.getSecretField(secretID, "username");
            String password = util.getSecretField(secretID, "password");
            String token    = util.getUserToken();

            System.out.println("User: '" + user + "'" );
            System.out.println("  pw: '" + password + "'");
            System.out.println("  token: '" + token + "'");
        } catch (java.lang.Exception e) {
            System.out.println("EXCEPTION");
            System.out.println(e.getMessage());
            System.out.println("STACK");
            e.printStackTrace();
        }
      }

}
```

API METHODS

getSecretField

Arguments

Secret ID

- ✓ Type: Integer
- ✓ Required: Yes
- ✓ The ID of the Secret to retrieve

Field Names

- ✓ Type: string parameters
- ✓ The name or names of the corresponding Secret Field

Return Type

String

Example

```
IUtilities util = new Utilities();
String password = util.getSecretField(8, "Password", "UserName");
```

Returns: “passwordvalue|username”

getSecretField – available in version 1.1.0

Arguments

SecretName

- Type: String
- Required: Yes
- The Name of the Secret to retrieve

FolderPath

- Type: string
- The full folder path where the Secret exists

Field Names

- Type: string parameters
- The name or names of the corresponding Secret Field

Return Type

String

Example

```
IUtilities util = new Utilities();
String password = util.getSecretField("SecretName", "Folder\SubFolder", "Password");
```

Returns: “passwordvalue”

getSecretFields

Arguments

Secret ID

- Type: Integer
- The ID of the Secret to retrieve

Field Names

- Type: String parameters

- ✓ The name or names of the corresponding Secret Field

Return Type

String

Example

```
IUtilities util = new Utilities();
String password = util.getSecretFields(8, ",", "Password", "UserName");
```

Returns: “passwordvalue,username”

getBase64EncodedFileAttachment

Arguments

Secret ID

- ✓ Type: Integer
- ✓ The Id of the Secret to retrieve

fieldName

- ✓ Type: String
- ✓ The field name of the file attachment on the Secret

filePath (optional)

- ✓ Type: String
- ✓ The file will always be returned as a base 64 encoded string, if a file path is specified, the API will write the file attachment to that path.

Return Type

String

Example

```
IUtilities util = new Utilities();
String password = util.getBase64EncodedFileAttachment(8, "secretfilefield", "C:\Temp\file.txt");
```

Returns: the file as a string, and also writes the file to the specified path.

getUserToken – Available in Version 1.1.0

Arguments

None

Return Type

String

Example

IUtilities util = new Utilities();
String password = util.getUserToken();

Returns: a web services token for the Application User that can be used with the SOAP API for extended functionality.

getUserRestToken – Available in Version 1.2.0

Arguments

None

Return Type

String

Example

IUtilities util = new Utilities();
String password = util.getUserRestToken();

Returns: a web services token for the Application User that can be used with the REST API for extended functionality.

Integrated .NET Configuration API

By installing the .NET Configuration API in the bin directory of a .NET web application a user can replace sensitive data stored in the file with symbolic tokens. The installation only requires running a .exe to install the Application Server API .dll file to the current directory (or other common directories such as c:\windows or c:\windows\system32).

Note Using the Integrated .NET Configuration API *does not* require any code changes or recompilation – it is simply adding a .dll to the server and making an edit to the configuration file. This implementation does not require a software engineer and can be done by infrastructure staff.

EXAMPLE

In a web configuration file <appSettings> node there is a hardcoded password to access a database:

```
<appSettings>
  <add key="productionpassword" value="SomeStrongPassword" />
</appSettings>
```

Using the .NET Configuration API, the .config file can be changed to:

```
  <appSettings>
    <add key="productionpassword" value="$$$\Servers\Web1\Password$$$" />
  </appSettings>
```

The user replaces the sensitive password with the full path, surrounded by tokens, to the specific Secret and Secret Field for the password stored in Secret Server. Then at runtime, the Application Server API will intercept the call to the configuration value and will replace with the corresponding password from Secret Server.

This allows configuration values to be replaced with symbolic tokens and does not require any source code changes to the application.