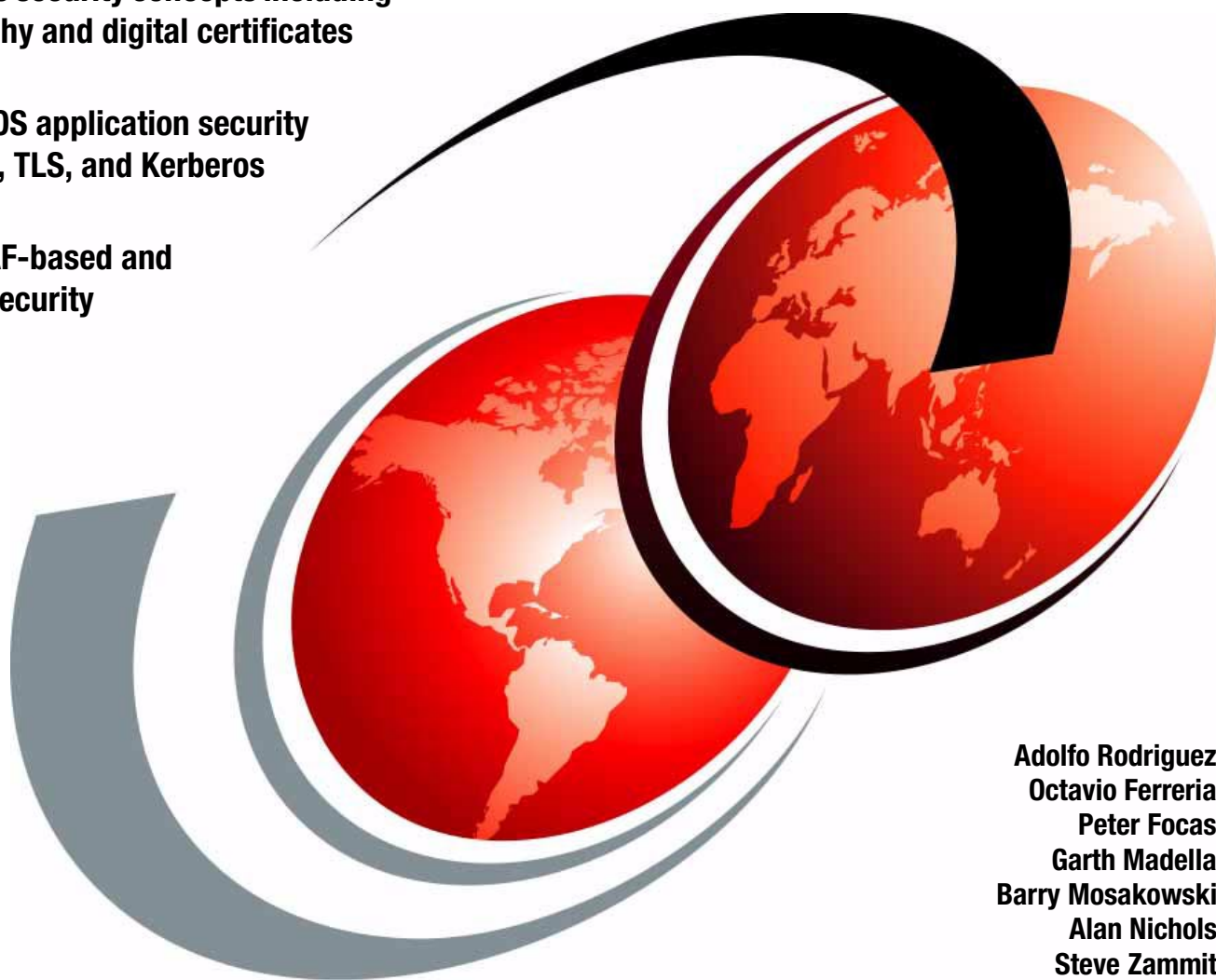


Communications Server for z/OS V1R2 TCP/IP Implementation Guide Volume 7: Security

Introduces security concepts including
cryptography and digital certificates

Details z/OS application security
using SSL, TLS, and Kerberos

Covers SAF-based and
network security



Adolfo Rodriguez
Octavio Ferreria
Peter Focas
Garth Madella
Barry Mosakowski
Alan Nichols
Steve Zammit

Redbooks



International Technical Support Organization

**Communications Server for z/OS V1R2 TCP/IP
Implementation Guide Volume 7: Security**

November 2002

Take Note! Before using this information and the product it supports, be sure to read the general information in “Notices” on page ix.

First Edition (November 2002)

This edition applies to Volume 1, Release 2 of Communications Server for z/OS IP.

Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept. HZ8 Building 662
P.O. Box 12195
Research Triangle Park, NC 27709-2195

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© Copyright International Business Machines Corporation 2002. All rights reserved.

Note to U.S Government Users - Documentation related to restricted rights - Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

Contents

Notices	ix
Trademarks	x
Preface	xi
The team that wrote this redbook	xii
Comments welcome	xiii
Part 1. Introduction	1
Chapter 1. Security in a networked world	3
1.1 Evolution of networking	4
1.2 Potential problems with electronic message exchange	5
1.2.1 The request is not really from your customer	5
1.2.2 The order could have been intercepted and read	6
1.2.3 The order could have been intercepted and altered	6
1.2.4 An order is received from your customer, but he denies sending it	7
Chapter 2. Basic cryptography	9
2.1 Secret key cryptography	10
2.2 Public key cryptography	11
2.2.1 Encryption	12
2.2.2 Authentication	12
2.2.3 Public key algorithms	13
2.2.4 Digital certificates	13
2.3 Performance issues of cryptosystems	17
2.4 Message integrity	18
2.4.1 Message digest (or “hash”)	18
2.4.2 Message authentication codes (MAC)	19
2.4.3 Digital signatures	20
Part 2. Securing z/OS with RACF	23
Chapter 3. UNIX System Services security	25
3.1 z/OS Security Server (RACF)	25
3.1.1 Identification and authentication	25
3.1.2 Alternatives to passwords	26
3.1.3 Checking authorization	26
3.1.4 Logging and reporting	27
3.1.5 RACF and z/OS UNIX System Services	28
3.2 Security in UNIX systems	28
3.2.1 Traditional UNIX security mechanisms	29
3.3 z/OS UNIX System Services security	31
3.3.1 UNIX level security	32
3.3.2 z/OS UNIX System Services level security	32
3.3.3 Why is z/OS UNIX System Services a more secure UNIX?	35
3.3.4 Access permission to HFS files and directories	36
3.3.5 Displaying files and directories	39
3.3.6 UID/GID assignment to a process	40
3.3.7 Defining UNIX System Services users	40

3.3.8	Default user	42
3.3.9	Superuser	43
3.3.10	Started task user IDs	46
3.3.11	FACILITY class profile BPX.SUPERUSER	47
3.3.12	FACILITY class profile BPX.DAEMON	48
3.3.13	Additional BPX.* FACILITY class profiles	49
3.3.14	Programs in the Hierarchical File System	50
3.3.15	z/OS UNIX kernel address space	52
3.3.16	z/OS UNIX security considerations for TCP/IP	52
3.3.17	IBM-supplied daemons	54
3.3.18	MVS sockets server applications	56
3.3.19	Summary	56
Chapter 4. TCP/IP stack resource access		59
4.1	TCP/IP stack access control	60
4.1.1	Stack Access overview	60
4.1.2	Example setup	61
4.1.3	Transport/stack affinity	63
4.2	Network access control	67
4.2.1	Network access control overview	67
4.2.2	Server considerations	68
4.2.3	Using NETSTAT for Network Access control	68
4.2.4	Working example of Network Access control	68
4.3	Port Access control	70
4.3.1	The PORT/PORTRANGE SAF keyword	71
4.3.2	SAF keyword on FTP or any other well-known PORTs	71
4.3.3	Using NETSTAT to display Port Access control	72
4.3.4	Scenarios using port access control	72
Chapter 5. Operations and administration security		75
5.1	z/OS VARY TCPIP command security	76
5.1.1	RACF profile details	76
5.1.2	VARY TCPIP command security scenario	77
5.2	TSO NETSTAT and UNIX onetstat command security	79
5.2.1	RACF profile details	79
5.2.2	NETSTAT security scenario	80
5.2.3	Further reading	81
Part 3. Network security		83
Chapter 6. Firewall concepts		85
6.1	General guidelines for implementing firewalls	86
6.2	Firewall categories	87
6.2.1	Packet filtering	87
6.2.2	Application-level gateway	88
6.3	z/OS Firewall Technologies	89
6.4	The demilitarized zone	89
Chapter 7. IPSec and virtual private networks (VPN)		91
7.1	IPSec	92
7.1.1	Security Associations	93
7.1.2	Transmitting data with IPSec	95
Chapter 8. Implementing IPSec with z/OS Firewall Technologies		99

8.1	Introduction	100
8.2	Firewall enhancements	100
8.3	Installation planning	101
8.4	Installation, configuration and operation	101
8.5	Interoperability considerations	101
8.6	Sample configuration files	102
8.7	RACF considerations	102
8.7.1	Configuring TCP/IP on the firewall host	103
8.8	Configuring and using the configuration server and client (GUI)	105
8.8.1	Simple configuration scenario	105
8.8.2	Configuring SSL	105
8.8.3	Configuring the configuration server (CFGSRV)	106
8.8.4	Setting up the configuration client on Windows	106
8.8.5	Accessing the configuration client (GUI)	107
8.8.6	Tunnel definition	109
8.8.7	FWTUNNL export file conversion from z/OS and AIX	118
8.8.8	On-demand dynamic tunnels scenario	119
Part 4	Application security	183
Chapter 9	Tools for application security	185
9.1	Secure Sockets Layer (SSL)	186
9.1.1	SSL protocol description	186
9.1.2	Certificates for SSL	188
9.1.3	System SSL	190
9.2	TLS protocol	191
9.3	Kerberos-based security system	192
9.3.1	Kerberos protocol overview	192
9.3.2	Inter-realm operation	197
9.3.3	Some assumptions	198
9.3.4	Kerberos implementation in z/OS	198
Chapter 10	Certificate management in z/OS	203
10.1	Digital certificates in z/OS	205
10.2	Digital certificate field formats	205
10.3	RACF RACDCERT command use	207
10.4	RACF keyrings	208
10.4.1	RACDCERT command security	209
10.4.2	RACDCERT command format	209
10.5	gskkyman command use	210
10.6	Client certificates	212
10.7	Server certificates	212
10.8	Self-signed certificates	213
10.9	Obtaining certificates	213
10.9.1	Self-signed certificates	213
10.9.2	Internal Certificate Authority (CA)	229
10.9.3	External Certificate Authority (CA)	234
10.10	Certificate locations example	247
10.10.1	RACF certificates	247
10.10.2	gskkyman HFS certificates	250
Chapter 11	File-related applications	255
11.1	z/OS FTP server	256
11.1.1	FTP using Transport Layer Security (TLS)	256

11.1.2	TLS/SSL scenarios	260
11.1.3	FTP using Kerberos	277
11.1.4	FTP and Kerberos scenario	283
11.2	z/OS TFTP server	287
11.3	z/OS NFS server	288
11.3.1	z/OS NFS security levels	289
11.3.2	Security information exchange between NFS client and server	290
11.3.3	Access to the HFS	294
11.3.4	Conclusion	296
Chapter 12.	TN3270 security	299
12.1	TN3270 SSL	300
12.1.1	TN3270 configuration parameters for SSL	301
12.1.2	Client authentication	302
12.1.3	TN3270 server SSL configuration scenarios.	303
12.2	Negotiated Telnet security	312
12.2.1	TN3270 server parameters for negotiated security.	314
12.2.2	TN3270 server configuration scenario	315
12.2.3	TN3270 client (HOD) negotiated Telnet configuration scenario	317
12.3	Express Logon Feature (ELF).	326
12.3.1	Two-tier network design	327
12.3.2	Three-tier network design	328
12.3.3	Implementing ELF in a two-tier design	330
12.3.4	Implementing ELF in a three-tier design.	336
Chapter 13.	UNIX remote execution applications	343
13.1	UNIX Telnet server security	344
13.1.1	Kerberized UNIX Telnet server support	345
13.2	UNIX System Services rlogind/rshd/rexecd	346
13.3	z/OS UNIX rshd Kerberos support	349
13.3.1	Implementing Kerberos on orshd	350
Chapter 14.	OMPRoute security	353
14.1	OSPF route update messages security	354
14.2	OMPRoute configuration.	354
14.2.1	The Area configuration statement.	354
14.2.2	The OSPF_Interface configuration statement.	355
Chapter 15.	Network management applications	357
15.1	z/OS SNMP.	358
15.1.1	SNMP security	359
15.2	z/OS Policy Agent	362
15.2.1	SSL with LDAP and Policy Agent	363
15.2.2	Considerations when opening an SSL connection	363
Chapter 16.	HTTP Server for z/OS	367
16.1	HTTP Server security	368
16.2	Server security structure	369
16.3	Setting up SAF control	369
16.4	How to protect resources	370
16.4.1	Access control directives.	371
16.4.2	Protection directives	375
16.5	Accessing back-end applications	377
16.6	SSL-related features in the IBM HTTP Server for z/OS	378

16.6.1	Encryption support	378
16.6.2	Global Server IDs	379
16.6.3	Crypto hardware support for SSL	379
16.7	SSL scenario	380
16.7.1	Server authentication	380
16.7.2	Client authentication	382
16.8	Associating a client certificate with a RACF user ID	387
16.8.1	RACF digital certificate support	387
16.8.2	Install and maintain digital certificates in RACF	387
16.8.3	Register a certificate using RACDCERT	389
16.8.4	Certificate self-registration with RACF	393
16.8.5	Certificate name filtering	393
16.9	Retrieving LDAP information	394
16.9.1	Configuring LDAP on IBM HTTP Server	395
16.9.2	How to use authentication information stored in LDAP	396
16.9.3	Creating user entries in the z/OS LDAP server	397
16.10	Conclusion	400
Chapter 17.	Utility applications	401
17.1	z/OS Lightweight Directory Access Protocol (LDAP)	402
17.1.1	Authentication with the z/OS LDAP server	402
17.1.2	Security of the directory	403
17.1.3	Using SSL communication	404
17.2	BIND-9 based DNS	409
17.2.1	TSIG	409
17.2.2	DNSSEC	411
17.2.3	Secure your DNS environment	413
17.3	Syslogd daemon	419
17.3.1	syslogd isolation	420
Part 5.	Appendixes	423
Appendix A.	VPN planning worksheets	425
Appendix B.	Sample RACF definitions	437
B.1	RACF settings for UNIX System Services	438
B.2	RACF settings for TCP/IP applications	438
B.2.1	RACF configuration for OS/390 UNIX level security	439
B.2.2	RACF definitions to control the use of the TCP/IP operator commands	440
B.3	Required RACF definitions to get Firewall Technologies started	440
B.4	RACF definition to manage certificate in RACF common keyring	441
Appendix C.	Default permissions for HFS files in z/OS UNIX	443
Appendix D.	Digital certificate formats supported by RACDCERT	445
Related publications		447
IBM Redbooks		447
Other resources		447
Referenced Web sites		448
How to get IBM Redbooks		448
IBM Redbooks collections		448
Index		449

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described 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 Licensing, IBM Corporation, North Castle Drive Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.


This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

IBM eServer™	MVS™	RS/6000®
Redbooks(logo)™ 	MVS/ESA™	S/390®
AIX®	Net.Data®	SecureWay®
AnyNet®	NetView®	SP™
Balance®	OpenEdition®	SP1®
CICS®	OS/2®	System/390®
DB2®	OS/390®	Tivoli®
FFST™	PAL®	VTAM®
IBM®	Parallel Sysplex®	WebSphere®
IBM.COM™	Perform™	z/OS™
IMS™	RACF®	zSeries™
Multiprise®	Redbooks™	

The following terms are trademarks of International Business Machines Corporation and Lotus Development Corporation in the United States, other countries, or both:

Lotus®	Notes®	Domino™
--------	--------	---------

The following terms are trademarks of other companies:

ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

C-bus is a trademark of Corollary, Inc. in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

SET, SET Secure Electronic Transaction, and the SET Logo are trademarks owned by SET Secure Electronic Transaction LLC.

Other company, product, and service names may be trademarks or service marks of others.

Preface

The Internet and enterprise-based networks have led to a rapidly increasing reliance upon TCP/IP implementations. The zSeries platform provides an environment upon which critical business applications flourish. The demands placed on these systems is ever-increasing and such demands require a solid, scalable, highly available, and highly performing operating system and TCP/IP component. z/OS and Communications Server for z/OS provide for such a requirement with a TCP/IP stack that is robust and rich in functionality. The *Communications Server for z/OS TCP/IP Implementation Guide* series provides a comprehensive, in-depth survey of CS for z/OS.

Volume 7 covers z/OS TCP/IP security issues. First, we begin with a survey of security issues and available methodologies for handling such issues as cryptography. We discuss SAF-based authorization techniques available on z/OS. We continue by presenting topics related to network security such as Firewall, VPN, and IPSec. Finally, we provide an in-depth look into application security including topics such as SSL, TLS, and Kerberos. We frame our discussion within application-specific chapters that cover everything from Telnet to FTP to DNS.

Because of the varied scope of CS for z/OS, this volume is not intended to cover all aspects of it. The main goal of this volume is to provide an insight into the different functions available for securing the CS for z/OS environment. For more information, including applications available with CS for z/OS IP, please refer to the other volumes in the series. These are:

- ▶ *Communications Server for z/OS V1R2 TCP/IP Implementation Guide Volume 1: Base and TN3270 Configuration*, SG24-5227
- ▶ *Communications Server for z/OS V1R2 TCP/IP Implementation Guide Volume 2: UNIX Applications*, SG24-5228
- ▶ *OS/390 eNetwork Communications Server for V2R7 TCP/IP Implementation Guide Volume 3: MVS Applications*, SG24-5229
- ▶ *Communications Server for z/OS V1R2 TCP/IP Implementation Guide Volume 4: Connectivity and Routing*, SG24-6516
- ▶ *Communications Server for z/OS V1R2 TCP/IP Implementation Guide Volume 5: Availability, Scalability, and Performance*, SG24-6517
- ▶ *Communications Server for z/OS V1R2 TCP/IP Implementation Guide Volume 6: Policy and Network Management*, SG24-6839

The team that wrote this redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization, Raleigh Center.

Adolfo Rodriguez is a Senior I/T Specialist at the International Technical Support Organization, Raleigh Center. He writes extensively and teaches IBM classes worldwide on all areas of TCP/IP. Before joining the ITSO, Adolfo worked in the design and development of CS for z/OS, in Research Triangle Park, NC. He holds a B.A. degree in Mathematics and B.S. and M.S. degrees in Computer Science from Duke University. He is currently pursuing the Ph.D. degree in Computer Science at Duke University, with a concentration on Networking Systems.

Octavio Ferreria is a Senior I/T Specialist in IBM Brazil. He has 17 years of experience in IBM software support. His areas of expertise include z/OS, VM, SNA, TCP/IP, LAN and WAN. For the last eight years, he has worked at the Area Program Support Group providing guidance and support to customers and designing networking solutions such as SNA to APPN migration and e-business solutions.

Peter Focas is a Network Systems Programmer in New Zealand. He has 12 years of experience in the SNA and TCP/IP networking field. His areas of expertise include the design and setup of SNA/SNI networks, configuration of secure TCP/IP servers using SSL/TLS, and digital certificate management.

Garth Madella is an Information Technology Specialist with IBM South Africa. He has 17 years of experience in the System/390 networking software field. He has worked with IBM for six years. His areas of expertise include VTAM, SNA, TCP/IP, and sysplex. He has written extensively on TCP/IP, sysplex, and Enterprise Extender issues.

Barry Mosakowski is a Software Engineer working at the IBM site in Research Triangle Park, North Carolina. He has eight years of experience in TCP/IP and SNA networking. He holds an MSE from Rensselaer Polytechnic Institute in Troy, NY. His areas of expertise include design, setup, and debugging of the Communication Server for z/OS TCP/IP to include the Telnet server, device drivers, socket applications, and Policy Agent.

Alan Nichols is an Independent Consultant living in Germany. He has 20 years of experience in MVS and z/OS and 10 years of UNIX and IP experience. He is currently working for last-level IP/USS support in T-Systems.

Steve Zammit is an Advisory I/T Specialist with the IBM Software Support Centre in Vancouver, Canada. Steve has 17 years of experience with IBM systems and networking, currently specializing in CS for z/OS TCP/IP. He holds a BSc degree in Applied Physics from Portsmouth Polytechnic, UK.

Thanks to the following people for their contributions to this project:

Bob Haimowitz, Jeanne Tucker, Margaret Ticknor, Tamikia Barrow, Gail Christensen, Linda Robinson
International Technical Support Organization, Raleigh Center

Jeff Hagggar, Bebe Isrel, Van Zimmerman, Jerry Stevens, Tom Moore, Dinakaran Joseph, Greg Callis, Andrew Arrowwood, David Yang, Dave Herr
Communications Server for z/OS Development, Research Triangle Park, NC

Comments welcome

Your comments are important to us!

We want our Redbooks to be as helpful as possible. Send us your comments about this or other Redbooks in one of the following ways:

- ▶ Use the online **Contact us** review redbook form found at:

ibm.com/redbooks

- ▶ Send your comments in an Internet note to:

redbook@us.ibm.com

- ▶ Mail your comments to:

IBM Corporation, International Technical Support Organization
Dept. HZ8 Building 662
P.O. Box 12195
Research Triangle Park, NC 27709-2195



Part 1

Introduction

This part introduces the issues involved with TCP/IP security. We discuss the need for security, the goals for the tools we use for security, and how the z/OS environment fits in with corporate-wide security policies.



Security in a networked world

This redbook is an implementation guide for securing access to the resources of a z/OS IP environment. We show how to configure the various operating system, TCP/IP stack, server and client components so that authorized users can access only those resources that they are permitted to access, but there is more to it than that. Once an authorized user has access to exchange messages with a server (or another user), there must be a means of securing those conversations such that the partners are sure of who they are talking to, that their conversation cannot be overheard, and that no messages can be altered or originated by a third party.

The word “security” means different things to different people. For example, the securing of a workstation behind locked doors, while probably a good idea, is not something that needs to be covered here. For the purposes of this book, it is assumed that a malicious user has access to your network, and is intent on doing damage. This assumption is based on the fact that the vast majority of security violations are from within an enterprise, accidental or otherwise. The types of damage an unauthorized user can inflict are fairly obvious, such as accessing or altering confidential data, or sending messages in the network pretending to be someone else. Some not-so-obvious damage could include tying up network resources by flooding messages into a network, or deliberately exploiting operating system or server software flaws to shut down parts of the system.

In this chapter, we present a brief history of how networks evolved, the potential security threats during a typical electronic message exchange over a network, and solutions to those threats. The subject of encryption is covered in Chapter 2, “Basic cryptography” on page 9 because it is pivotal to other technologies and processes that will be presented throughout this book.

The following chapters deal with the subject of TCP/IP security in z/OS based computers. The unique security environment of the z/OS platform incorporating the Security Access Facility (SAF) provides many ways to secure the resources of your TCP/IP system. The standard SAF product for z/OS is RACF and that will be used for all the examples in this book.

1.1 Evolution of networking

At the beginning of the computing era, computers stood alone, with input devices limited to card readers, disks, and tapes. People would input information via punched cards, programs would process it, and printers would print it. As more and more companies deployed computers, it became obvious that computers should start talking to each other to exchange information so that it didn't need to be manually entered.

Early networks were set up with proprietary hardware and software running over private links. As more and more computers were connected together, establishing a link between each one became more costly both in setup and maintenance.

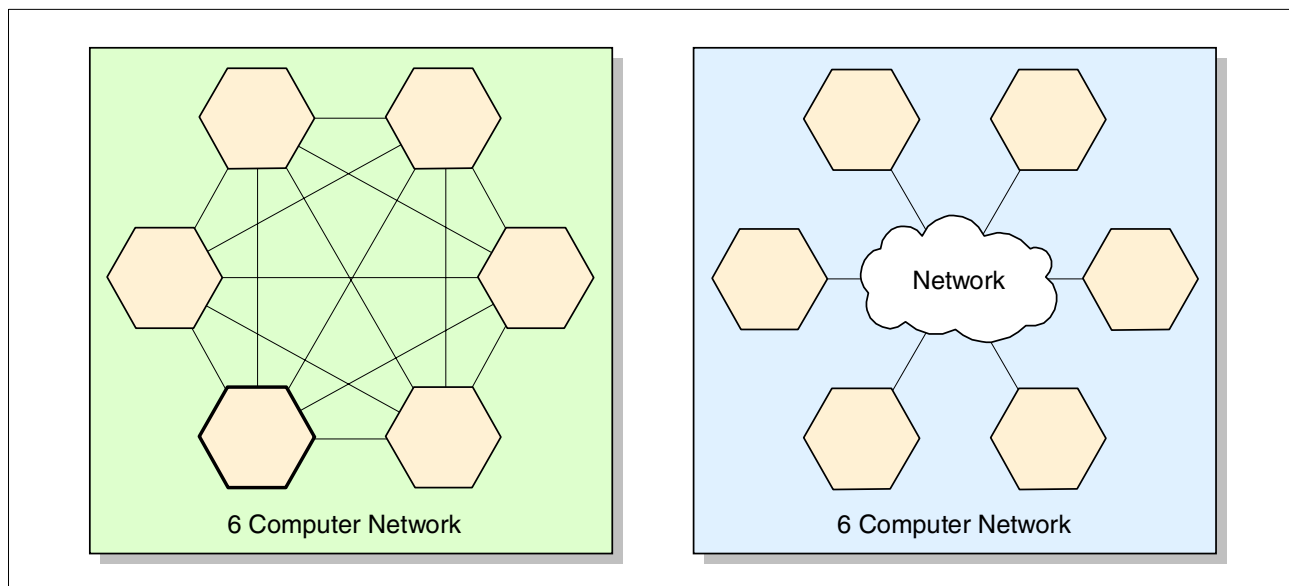


Figure 1-1 Individual connections between computers versus network attachment

As shown in Figure 1-1, to connect six computers together so that they all have a single link to each other would take 15 lines ($5 + 4 + 3 + 2 + 1$). To connect 10 computers the same way would take 45 lines ($9 + 8 + 7 \dots$). To connect 100 computers together....well, you get the point. At some stage, it is going to be more economical to connect your computer to a network that everyone shares. An early example of this type of network is X.25, a packet switching network. Using X.25, computers can set up a *virtual channel* between any two computers over a shared network, with only one physical attachment to that network. X.25 has the advantage that the protocol is implemented by almost all computer vendors, and it is still used today.

Back in the early 1970s, another type of network was in its infancy. The US military started a research program to design an open-architecture network with the capability of allowing computers to exchange messages over multiple packet-switched networks. This network was named ARPANET (Advanced Research Project Agency Network) and this network eventually transformed into the Internet.

The development of the Internet and its related protocols is, without doubt, the cause of the explosive growth of networked computers around the globe. Who would have thought it possible, only 15 years ago, that we would have users scattered around the world all connected together to play music and movies, exchange messages, do shopping, access libraries, book travel, and more? The protocols developed for use on the Internet also have relevance on a private enterprise network. They are, after all, designed to allow computers and users to talk to each other and that can occur over a public or private network. A network

using Internet protocols but not generally available to the public is termed an *intranet*. From a security perspective, Internet-connected computers and computers on a private intranet have very similar security issues, unless you have total trust in all of the users of your private network (if you do, you don't need this book for the most part).

The Internet infrastructure is now so globally entrenched that it makes sense for individuals and companies to use it for exchanging electronic messages, rather than setting up private networks (perhaps with leased or switched telecommunication lines). A message being exchanged between two users on the Internet will almost always flow through third-party computers (routers) on the way. The owners of these routers can in no way be considered trustworthy, and when those messages deal with the transfer of financial securities or confidential information of any kind, the security exposure should be obvious.

1.2 Potential problems with electronic message exchange

Let's take an example of an electronic message exchange, and examine what potential security problems could occur:

You are the manager of a stock broking company and you want an online stock trading system for your clients. You want your clients to be able to log in to your system and place buy and sell orders. You imagine a scenario where one of your clients has logged in to your system, to confidentially request the purchase of one million shares in XYZ Corporation. Your system has to have an extremely secure way of checking that the request is from your client. What if some hacker had gained access to the system? Furthermore, the request must be extremely confidential. If the purchase request had been intercepted and read, other people may know what your client is up to. What if the request had been intercepted and altered? Your customer may have only been asking for 1000 shares and now you're going to buy 1 million. What would happen if you bought the shares and then the customer denies ever having requested them? The problems seem insurmountable.

You identify the following potential problems involved with this message exchange:

- ▶ The request is not really from your customer.
- ▶ An order is received from your customer that could have been intercepted and read.
- ▶ An order is received from your customer that could have been intercepted and altered.
- ▶ An order is received from your customer, but after you buy the stock, he denies sending it.

Now we discuss what can be done about each of these problems.

1.2.1 The request is not really from your customer

What is needed here is some way for you to ensure that the customer is who he says he is. In some cases, this must involve some sort of shared secret, such as a password. This is called *user authentication*.

When you authenticate something, you are verifying its identity or validity. Traditional user authentication normally involves a user supplying a password that must match your record of the password. When a password is sent across a network, it is possible that someone eavesdropping on the network could see it. The eavesdropper could then log in, using the password discovered to gain access to your customer's account. One way of resolving this problem is to encrypt the password when it is sent across the network.

Another way to protect passwords from exposure in a network is to not send the password at all. There are a number of protocols available to authenticate a user, so that the password is not sent over the network. One example of this type of authentication scheme is the

Challenge Authentication Protocol, where the two parties of the conversation share a password. Lets assume the customer sent a login request to the brokers trading system server quoting his login ID. The server would then send a random number (the *challenge*) back to the customer's client software (and would store it locally for later use). The customer's client software gets the challenge and alters it in some way, by some complex formula that involves the shared password, and sends it back to the trading system server. When the server receives the challenge back, now modified by the formula and the shared password, the server computes its own version using the locally stored user password and the random number originally sent. This must match what was returned by the customer.

In this protocol, no passwords actually flow on the network, but there are a number of problems. One problem is how do you get the copy of the shared password securely to all people with whom you will communicate? If you already had a secure channel to distribute passwords, wouldn't you use that for the trading requests? In a system that is accessed only by local clients, a shared password can be distributed manually, but in a public system you must have a more efficient method of key distribution. This problem is addressed and discussed in 2.2, "Public key cryptography" on page 11.

1.2.2 The order could have been intercepted and read

Assume you have some way of knowing, for sure, that the order you have received was originated by your customer. How do you know that the order has not been read by anyone other than the two parties involved? You can't be sure how many computers and links it has been across, and you don't know whether any intermediate link in the network has cached the message or logged it in any way, so what can you do? The sender must alter the message so that its meaning is hidden to unauthorized parties, a process known as *encryption*. In a simple encryption scheme, you may decide to replace every character in the message with another, unique character replacement. This scheme could easily be broken by a number of methods, depending on the type of information being encrypted. For instance, if the encryption was being applied to English language text, the most common character field would probably be the letter "E", followed by "T" and so on. There are encryption schemes that are virtually unbreakable, however.

Encryption schemes are fundamental to the operation of many security protocols and processes. For a discussion of encryption, see "Basic cryptography" on page 9.

1.2.3 The order could have been intercepted and altered

How do you know, when you receive a message, that the contents of the message haven't been intercepted and regenerated while the message was passing through an intermediate network node? This could result in dire consequences, depending on what the message is. The message "I wish to buy 100 shares in XYZ Corporation" is quite different in meaning from "I wish to sell 100 shares in XYZ Corporation", although only a few characters have been altered. Messages traveling over electrical cables are always subject to some form of electromagnetic interference. In most cases, the protocol managing the transfer of data over a wire will append a Cyclic Redundancy Check (CRC) to a message. In a simple CRC, this might be calculated by adding all the 8-bit bytes in a message, and then taking the low-order 16 bits as the CRC. If interference is present on the cable, the CRC *may* not add up, and the protocol will signal that a retransmission is necessary. CRCs are normally calculated and inserted into a network frame by hardware.

If we are talking about a deliberate alteration of a message (and we are), then a CRC is useless. The attacker would just intercept the message, send it out to the network and the network adapter would regenerate the new CRC according to the message that was altered. What is needed, is some form of *message authentication*.

Put simply, a message authentication process takes a message block, or stream, and mathematically summarizes the bits in the message to produce a fixed length *message digest* that represents that message. No two messages should produce the same message digest, or at least, it should be computationally infeasible to find two that do. This message digest is normally appended to the original message, and transmitted along with it, to the destination. If the original message is altered, when the receiver recalculates the message digest, it will differ from the one in the message. This does not guard against someone intercepting the message, altering it, recalculating the digest and replacing the original digest and sending it along. That is why digests are often encrypted, which is then termed a *message authentication code* (MAC).

If the encryption process is by a private key (nobody else knows the private key) then the MAC becomes a *digital signature*. A digital signature proves that one party, and one party alone, could have originated a particular message. At this point you may be wondering why we would bother with message hashes, digital signatures, CRCs, etc. Why not just encrypt the whole message and be done with it? If the message was intercepted and altered, the decryption process would yield rubbish and the receiver would know the message should be retransmitted.

The reason is that encryption is relatively expensive in processor use. It is a lot quicker to encrypt just a small part of a message (the appended message digest) than the whole message itself. This is an important point that explains why a lot of security issues are not just handled by strong encryption immediately. It also explains why a MAC (encrypted message digest) would be appended to an unencrypted message.

Authentication of messages is discussed in 2.4, “Message integrity” on page 18.

1.2.4 An order is received from your customer, but he denies sending it

In a normal day-to-day transaction, such as cashing a check at the bank, you provide a signature that goes a very long way to proving who you are and that you wanted this transaction to be performed. In an electronic world, where malicious messages can be injected into the network by any 12-year old with a computer and an attitude problem, it is important to know that a particular message came from a particular person. For instance, in our example, if the stock broker’s trading system decided to buy the million shares in XYZ Corporation on behalf of the client, and then the value of the stock plummeted in value, the client might decide a clever thing to do would be to deny having requested the stock in the first place. This is obviously a huge problem for the stock broker. What is needed is some method where the sender of a message cannot deny having sent it. This requirement is called *non-repudiation*.

This poses an interesting problem. How can you construct an electronic message so that only you could have constructed it? Once again, the answer lies in encryption. Briefly, encryption comes in two broad categories depending on whether the encryption key for both parties is the same (*symmetric key*) or different (*asymmetric key*). If the key is the same on both sides of the conversation, either side could have created, and encrypted, an individual message. If the key is different, then only one side could encrypt a message in a particular way.

For a detailed discussion on non-repudiation, please see 2.4, “Message integrity” on page 18 and in particular 2.4.3, “Digital signatures” on page 20.



Basic cryptography

The word “cryptography” has its roots in Greek, and means “secret writing”. One of the earliest uses of cryptography was for protecting military communications. In ancient times, a human messenger would be dispatched with a military order. If that messenger were caught, the message could be read by the enemy. A method had to be used to hide the meaning of the message from an interceptor but still allow the intended recipient to understand it.

The message (plain text) to be conveyed would have to be encrypted by some formula (the *cipher*). The cipher normally has, as its inputs, the message to be encrypted and a *key*. By using a key, the cipher itself can be public knowledge but the key is kept (hopefully) private between the communicating parties. The text that is produced by the cipher is the *ciphertext*. The decryption process takes the ciphertext, runs it through the decryption cipher, with the key, and produces the plain text again.

Cryptography has more uses than ensuring privacy through encrypting a message. Other uses for cryptography are to provide message integrity through the use of encrypted message hashes, and non-repudiation so that a sender cannot deny having sent a particular message. To ensure privacy, integrity and non-repudiation in non-secure networks, cryptographic procedures need to be used.

Today, two distinct classes of cryptographic algorithms are in use:

- ▶ Secret key (or symmetric key)
- ▶ Public key (or asymmetric key)

They are fundamentally different in how they work, and thus in where they are used.

This chapter covers secret key and public key cryptography. How cryptography can aid in asserting message integrity (ensuring a message has not been altered in transit) is explained, and so are digital signatures that can prove that a holder of a particular private key originated a message. Digital certificates are covered, along with their role in the secure use of public key cryptography.

These are the basic building blocks for securing transactions over the Internet or some other untrusted network.

2.1 Secret key cryptography

Secret key cryptography is so-called because the key used to encrypt the message must be kept secret from everyone but the two communicating parties. Ensuring a key is secret seems obvious but is not the case in public key systems, described in 2.2, “Public key cryptography” on page 11. Another name for secret key encryption is symmetric encryption, so called because the same key that is used to encrypt the data is also used to decrypt the data and recover the clear text, as shown in Figure 2-1.

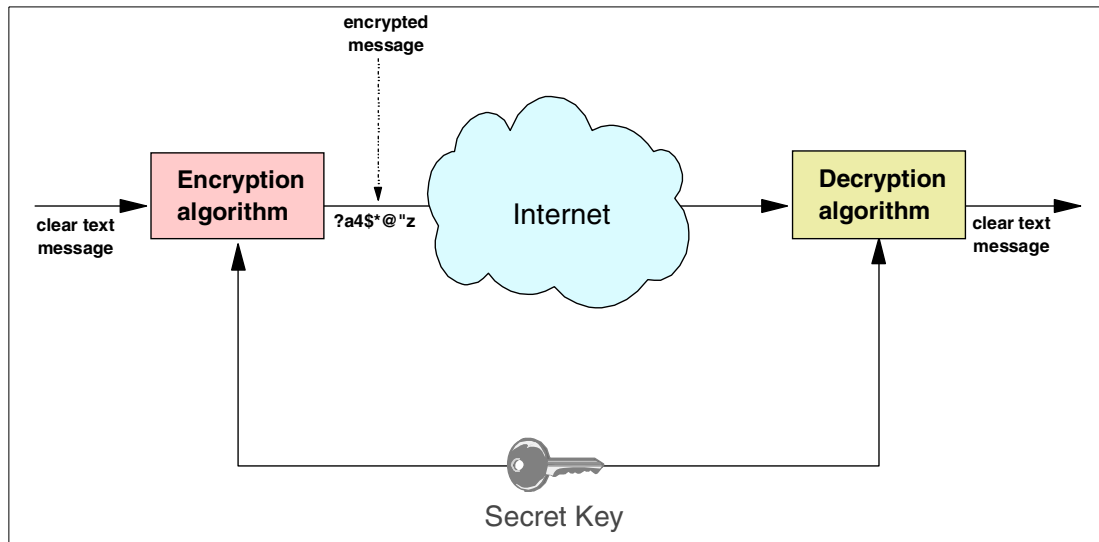


Figure 2-1 Symmetric encryption and decryption: using the same key

Symmetric algorithms are usually efficient in terms of processing power, so they are ideal for encryption of bulk data. However, they have one major drawback, which is key management. The sender and receiver on any secure connection must share the same key; in a large network where thousands of users may need to communicate securely, it is extremely difficult to manage the distribution of keys so as not to compromise the integrity of any one of them. Public key encryption, described in 2.2, “Public key cryptography” on page 11, can be used to exchange secret keys securely, and from then onward, the conversation can use the faster secret key encryption.

Frequently used symmetric algorithms include:

- DES** Data Encryption Standard. Developed in the 1970s by IBM scientists, it uses a 56-bit key. Stronger versions called Triple-DES have been developed that use three operations in sequence: “2-key Triple DES” encrypts with key 1, decrypts with key 2, and encrypts again with key 1. The effective key length is 112 bits. “3-key Triple-DES” encrypts with key 1, decrypts with key 2, and encrypts again with key 3. The effective key length is 168 bits.
- CDMF** Commercial Data Masking Facility. This is a version of the DES algorithm approved for use outside the U.S. and Canada (in times when export control was an issue). It uses 56-bit keys, but 16 bits of the key are known, so the effective key length is 40 bits.
- RC2** Developed by Ron Rivest for RSA Data Security, Inc., RC2 is a block cipher with variable key lengths operating on 8-byte blocks. Key lengths of 40, 56, 64, and 128 bits are in use.

- RC4** Developed by Ron Rivest for RSA Data Security, Inc., RC4 is a stream cipher operating on a bit stream. Key lengths of 40 bits, 56 bits, 64 bits, and 128 bits are in use. The RC4 algorithm always uses 128-bit keys; the shorter key lengths are achieved by “salting” the key with a known, non-secret random string.
- AES** As a result of a contest for a follow-on standard to DES held by the National Institute for Standards and Technology (NIST), the Rijndael algorithm was selected. This is a block cipher created by Joan Daemen and Vincent Rijmen with variable block length (up to 256 bits) and variable key length (up to 256 bits).
- IDEA** The International Data Encryption Algorithm was developed by James Massey and Xuejia Lai at ETH in Zurich. It uses a 128-bit key and is faster than triple DES.

DES is probably the most scrutinized encryption algorithm in the world. Much work has been done to find ways to break DES, notably by Biham and Shamir, but also by others. However, a way to break DES with appreciably less effort than a brute-force attack (breaking the cipher by trying every possible key) has not been found.

Both RC2 and RC4 are proprietary, confidential algorithms that have never been published. They have been examined by a number of scientists under non-disclosure agreements.

With all the ciphers listed above, it can be assumed that a brute-force attack is the only means of breaking the cipher. Therefore, the work factor depends on the length of the key. If the key length is n bits, the work factor is proportional to $2^{(n-1)}$.

Today, a key length of 56 bits is generally only seen as sufficiently secure for applications that do not involve significant amounts of money or critically secret data. If specialized hardware is built (such as the machine built by John Gilmore and Paul Kocher for the Electronic Frontier Foundation), the time needed for a brute-force attack can be reduced to about 100 hours or less (see *Cracking DES: Secrets of Encryption Research, Wiretap Politics & Chip Design*, by Electronic Frontier Foundation, John Gilmore (Editor), 1988). Key lengths of 112 bits and above are seen as unbreakable for many years to come, since the work factor rises exponentially with the size of the key.

2.2 Public key cryptography

Public key cryptography implements encryption and decryption using two different keys, which is why it is also termed *asymmetric encryption*. These two keys are known as a public key and a private key.

The beauty of asymmetric algorithms is that they are not subject to the key management issues that beset symmetric algorithms. Your public key is freely available to anyone, and if someone wants to send you a message he or she encrypts it using that key. Only you can understand the message, because only you have the private key.

Important: Public and private keys, if implemented in a reversible scheme such as RSA, (described next) yield extremely important properties:

- ▶ If the public key is used to encrypt the data, the private key must be used to recover the clear text.
- ▶ If the private key is used to encrypt the data, the public key must be used to recover the clear text.

2.2.1 Encryption

Figure 2-2 shows an exchange where one party (on the left) uses the second party's public key to encrypt a message.

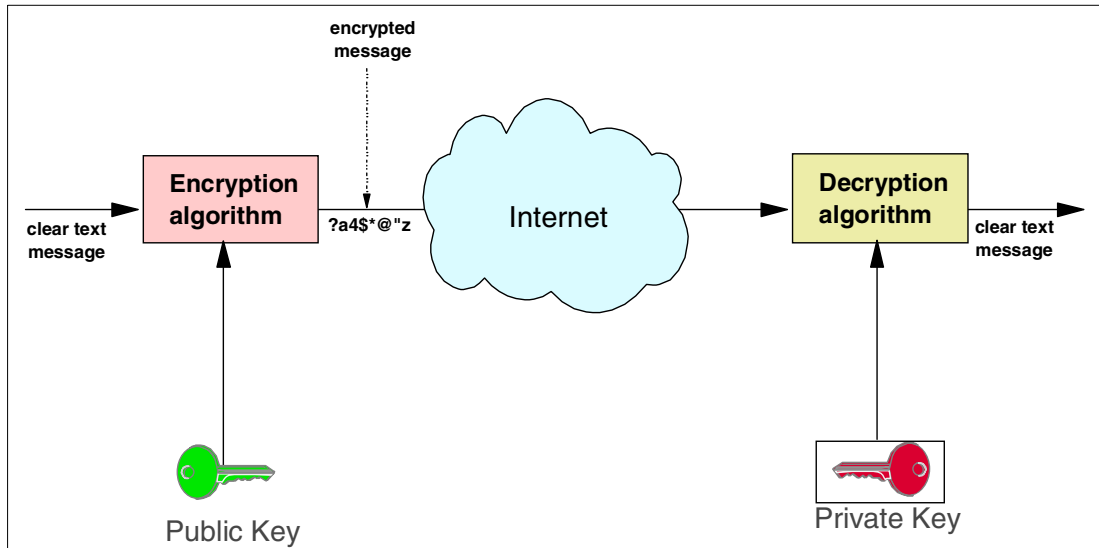


Figure 2-2 Public-key cryptography: encryption using a public key

The ciphertext created by this encryption process is only decipherable by using the private key, which in turn is only known by the second party. There is no way for any other party to decipher this message. This type of encryption is used when you want the receiver to be the only person capable of understanding the message. This message flow can also be used to securely exchange a secret key between the conversation partners so that the faster secret key encryption can be used instead of public key.

2.2.2 Authentication

Asymmetric keys are also very useful for authentication. Look at Figure 2-3. What happens if you encrypt a message using your own private key?

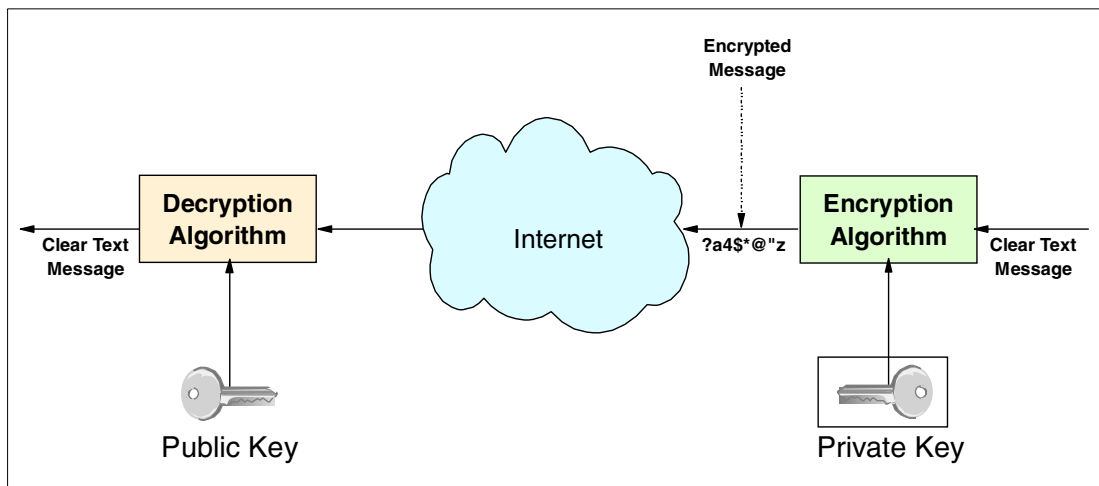


Figure 2-3 Public-key cryptography: encryption using a private key results in authentication

As stated earlier, this would indicate anybody with access to your public key (and that should be anyone) would be able to decipher the message. This type of encryption, therefore, is obviously of no use to hide a message. By encrypting a message with your private key, *a receiver must use your public key to decipher it*, and that is the point, this proves that the message could *only* have come from you.

2.2.3 Public key algorithms

Asymmetric encryption algorithms, commonly called Public Key Cryptography Standards (PKCS), are based on mathematical algorithms. The basic idea is to find a mathematical problem that is very hard to solve. The algorithm in most widespread use today is RSA. However, some companies have begun to implement public-key cryptosystems based on so-called “elliptic curve” algorithms. With the growing proliferation of IPSec, the Diffie-Hellman algorithm is gaining popularity. A brief overview of all three methods follows:

- RSA** Invented in 1977 by Rivest, Shamir, and Adleman (who formed RSA Data Security Inc.). The idea behind RSA is that integer factorization of very large numbers is extremely hard to do. Key lengths of public and private keys are typically 512 bits, 768 bits, 1024 bits, or 2048 bits. The work factor for RSA with respect to key length is sub-exponential, which means the effort does not rise exponentially with the number of key bits. It is roughly $2^{(0.3 \cdot n)}$.
- Elliptic Curve** Public-key cryptosystems based on elliptic curves use a variation of the mathematical problem of finding discrete logarithms. It has been stated that an elliptic curve cryptosystem implemented over a 160-bit field has roughly the same resistance to attack as RSA with a 1024-bit key length. Properly chosen elliptic curve cryptosystems have an exponential work factor (which explains why the key length is so much smaller). Elliptic curve cryptosystems are now standardized by FIPS PUB 186-2, the digital signature standard (January 2000).
- Diffie-Hellman** W. Diffie and M.E. Hellman, the inventors of public key cryptography, published this algorithm in 1976. The mathematical problem behind Diffie-Hellman is computing a discrete logarithm. Both parties have a public-private key pair each; they are collectively generating a key only known to them. Each party uses its own private key and the public key of the other party in the key generation process. Diffie-Hellman public keys are often called *shares*.

2.2.4 Digital certificates

Digital certificates are used to publish a public key with a certainty that the public key is genuine, according to the Certificate Authority that digitally signs the certificate. First we discuss what can happen when a public key is used for communication, and that key is not genuine. We then cover what can be done about authenticating a public key by using digital certificates.

How can I trust a published public key?

If we want to communicate with XYZ Corporation, and we have found a public key published on the Internet, how could we use that public key? The two uses of another person’s or entity’s public key are:

1. To decrypt a message originating from that person, who has encrypted with his private key
2. To encrypt a message to be sent to that person, so that only he can decrypt it, with his private key

As mentioned, we have found XYZ Corporation's public key on the Internet. How do we know it's genuine? A malicious third party could have put his own public key on the Internet and now could intercept all communications from you to XYZ Corporation, acting as a sort of "relay" on the way.

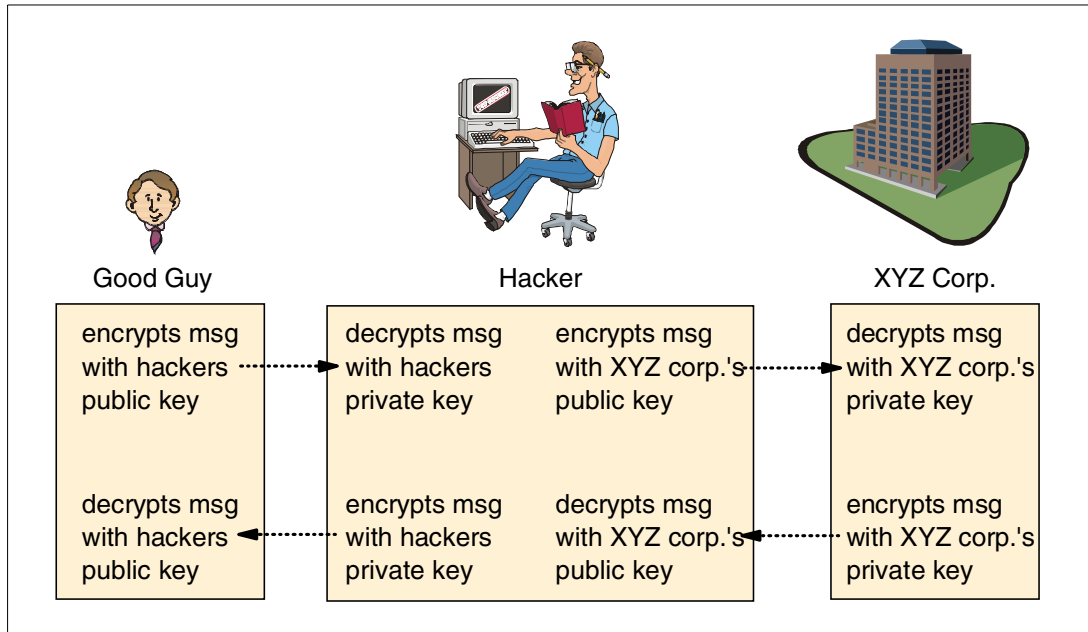


Figure 2-4 Scenario where public key being used by "good guy" is really a hacker's key

In Figure 2-4, the "good guy" assumes that he has obtained a public key for XYZ Corporation from the Internet, but in reality, it was a hacker's public key. We further assume that the hacker has some way of removing your messages from the network, and injecting his own. The last assumption is that we are using XYZ's public key (or at least we think we are) to encrypt messages to XYZ Corp and the response will be encrypted by XYZ Corp with its private key.

You can see what could happen when a public key is used for communication when you don't know if it's genuine. Your messages to XYZ Corporation will be encrypted using the hacker's public key, because you thought it was XYZ Corp's public key. The hacker then uses his private key to decrypt the message, make any changes he feels is necessary, and then encrypt the message with XYZ Corp's real public key. When XYZ Corp receives the message, it will decrypt using its private key, process the message and send a message back, encrypting with its own private key. The hacker then receives the response and decrypts using XYZ Corp's public key, makes more changes, if necessary, and encrypts with his own (hacker's) private key. Lastly, you receive the hacker's message, decrypt with what you think is XYZ Corp's public key, and now your communication to XYZ has been totally compromised.

The problem of securely storing and retrieving public keys is dealt with by what is known as a Public Key Infrastructure (PKI), discussed next. A good reference work on PKI can be found in the redbook *Deploying a Public Key Infrastructure*, SG24-5512 at <http://www.redbooks.ibm.com>.

Public Key Infrastructure

A Public Key Infrastructure (PKI) offers the basis for practical usage of public key cryptography. A PKI defines the rules and relationships for certificates and Certificate Authorities (CAs). It defines the fields that can or must be in a certificate, the requirements and constraints for a CA in issuing certificates, and how certificate revocation is handled.

When using a PKI, the user must be confident that the obtained public key belongs to the correct remote person (or system) with which the digital signature mechanism is to be used. This confidence is obtained through the use of public key digital certificates. A digital certificate is analogous to a passport: the passport certifies the bearer's identity, address, and citizenship. The concepts behind passports and other identification documents (for instance, drivers' licenses) are very similar to those that are used for digital certificates.

Passports are issued by a trusted authority, such as a government passport office. A passport will not be issued unless the person who requests it has proven their identity and citizenship to the authority. Specialized equipment is used in the creation of passports to make it very difficult to alter the information in it or to forge a passport altogether. Other authorities, for instance the border police in other countries, can verify a passport's authenticity. If they trust the authority that issued the document, they implicitly trust the passport.

A digital certificate serves two purposes: it establishes the owner's identity and it makes the owner's public key available. Similar to a passport, a certificate must be issued by a trusted authority, the CA, and, like a passport, it is issued only for a limited time. When its expiration date has passed, it must be replaced.

Trust is a very important concept in passports, as well as in digital certificates. In the same way as, for instance, a passport issued by the governments of some countries, even if recognized to be authentic, will probably not be trusted by the U.S. authorities, each organization or user has to determine whether a CA can be accepted as trustworthy.

As an example, a company might want to issue digital certificates for its own employees from its own Certificate Authority. This could ensure that only authorized employees are issued certificates, as opposed to certificates being obtained from other sources such as a commercial entity such as VeriSign.

The information about the certificate owner's identity is stored in a format that follows RFC 2253 and the X.520 recommendation, for instance: CN=Ulrich Boche, O=IBM Corporation. The complete information is called the owner's distinguished name (DN). The owner's distinguished name and public key and the CA's distinguished name are digitally signed by the CA. That is, a message digest is calculated from the distinguished names and the public key. This message digest is encrypted with the private key of the CA.

Figure 2-5 on page 16 shows a simplified layout of a digital certificate.

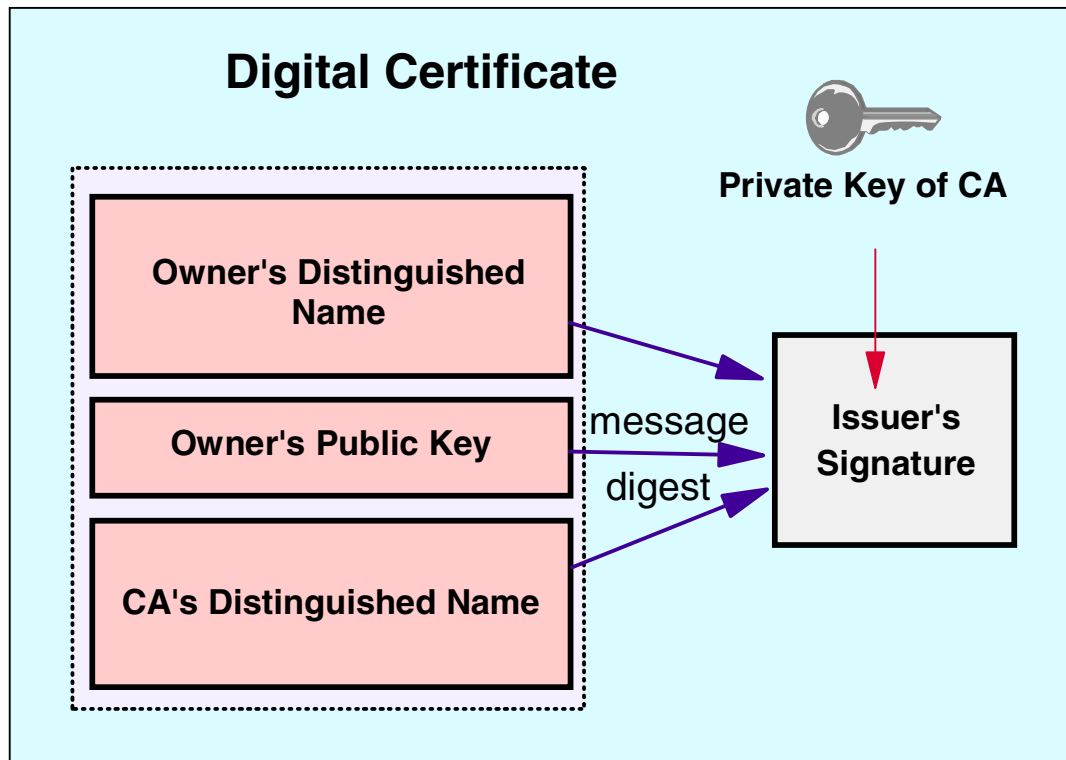


Figure 2-5 Simplified layout of a digital certificate

The digital signature of the CA serves the same purpose as the special measures taken for the security of passports, such as laminating pages with plastic material. It allows others to verify the authenticity of the certificate. Using the public key of the CA, the message digest can be decrypted. The message digest can be recreated. If it is identical to the decrypted message digest, the certificate is authentic.

Security considerations for certificates

If I send my certificate with my public key in it to someone else, what keeps this person from misusing my certificate and posing as myself? The answer is: my private key.

A certificate alone can never be proof of anyone's identity. The certificate just allows the identity of the certificate owner to be verified by providing the public key that is needed to check the certificate owner's digital signature. Therefore, the certificate owner must protect the private key that matches the public key in the certificate. If the private key is stolen, the thief can pose as the legitimate owner of the certificate. Without the private key, a certificate cannot be misused.

An application that authenticates the owner of a certificate cannot accept just the certificate. A message signed by the certificate owner should accompany the certificate. This message should use elements such as sequence numbers, time stamps, challenge-response protocols, or other data that allow the authenticating application to verify that the message is a "fresh" signature from the certificate owner and not a replayed message from an impostor.

Certificate Authorities and trust hierarchies

Before we discuss what is termed a “certificate hierarchy”, let’s look at an analogous example of trusted hierarchies. If you were selling a car, and a buyer asked you if it was OK to pay by personal check, then you would have to decide whether you trusted the buyer or not. If you do, end of story, the car is sold. If you do not trust him, you may ask someone whom you both trust to countersign the check.

In digital certificate trust hierarchies, similar considerations to the car-buying example apply. In the end, it boils down to the fact that you have to trust somebody. You will have digital certificates in your database, and those certificates will either be set to “trusted” or “untrusted” status. A Certificate Authority (CA) is a company that is considered trustworthy, and produces digital certificates for other individuals and companies (called “subjects”) bearing that subject’s public key. This certificate is signed with a message hash that is encrypted using the CA’s private key. To verify that the certificate is authentic, the receiver needs the public key of the CA that issued the certificate.

Most Web browsers come preconfigured with the public keys of common CAs (such as VeriSign). However, if the user does not have the public key of the CA that signed the certificate, an additional certificate would be needed in order to obtain that public key. In general, a chain of multiple certificates may be required, comprising a certificate of the public key owner signed by a CA, and possibly additional certificates of CAs signed by other CAs. Many applications that send a subject’s certificate to a receiver send not only just that certificate, but also all the CA certificates necessary to verify the certificate up to the root.

Obtaining and storing certificates

As we have discussed, certificates are issued by a CA. If you do not want to use a CA, you can use utilities to issue your own certificates. These are called “self-signed” certificates and will only be accepted by people who trust you. If you use an external Certificate Authority, you request certificates by visiting the CA’s Web site. After verifying the validity of the request, the CA sends back the certificate in an e-mail message or allows it to be downloaded.

In the case of obtaining a certificate for a server, whether you use a self-signed or external CA signed certificate is dependent on the server environment. In an intranet environment, it is generally appropriate to use self-signed certificates. In an environment where external users are accessing the server over the Internet, it is usually advisable to acquire a server certificate from a well-known CA, because the steps needed to import a self-signed certificate might seem obscure, and most users will not have the ability to discern whether the action they are performing is of trivial consequence or not. It should also be noted that a root CA certificate received over an untrusted channel, such as the Internet, does not deserve any kind of trust.

Certificate management in OS/390 and z/OS

To manage certificates on an OS/390 system, you can use either the UNIX program gskkyman to create and manage certificates, or you can use the RACF database and RACDCERT command. This topic is covered in detail in Appendix 10, “Certificate management in z/OS” on page 203.

2.3 Performance issues of cryptosystems

Elliptic curve cryptosystems are said to have performance advantages over RSA in decryption and signing. While the possible differences in performance between the asymmetric algorithms are somewhere in the range of a factor of 10, the performance differential between symmetric and asymmetric cryptosystems is far more dramatic.

For instance, it takes about 1000 times as long to encrypt the same data with RSA (an asymmetric algorithm) as it takes with DES (a symmetric algorithm), and implementing both algorithms in hardware does not change the odds in favor of RSA.

As a consequence of these performance issues, the encryption of bulk data is usually performed using a symmetric cryptosystem, while asymmetric cryptosystems are used for electronic signatures and in the exchange of key material for secret-key cryptosystems. With these applications, only relatively small amounts of data need to be encrypted and decrypted, and the performance issues of public key systems are less important.

2.4 Message integrity

Message integrity is the ability to assert that a message received has not been altered in any way from the time that it was sent. In a networked environment, a message could have been altered by a third party intercepting it, or by some other means, such as electromagnetic interference (although in the latter case the transmission protocol normally handles a retransmission). To provide message integrity, you provide a message digest along with the text of your message. Note that the message being authenticated may or may not also be encrypted.

2.4.1 Message digest (or “hash”)

A message digest algorithm takes a message as input, and produces a small, fixed length “digest” string (usually 128 or 160 bits) often referred to as a *hash*. This hash can be thought of as a mathematical summary of a message. There are two important things to note about a message digest algorithm.

- ▶ The algorithm is a *one-way* function. This means that there is absolutely no way you can recover a message, given the hash of that message.
- ▶ It should be computationally infeasible to produce another message that would produce the same message digest as another message.

Figure 2-6 is a graphical representation of appending a message digest to a message. When a message digest is appended to a message en-route to its destination, the message cannot be tampered with, because a recalculation of the hash at the receiver's end will show the message digest received is invalid.

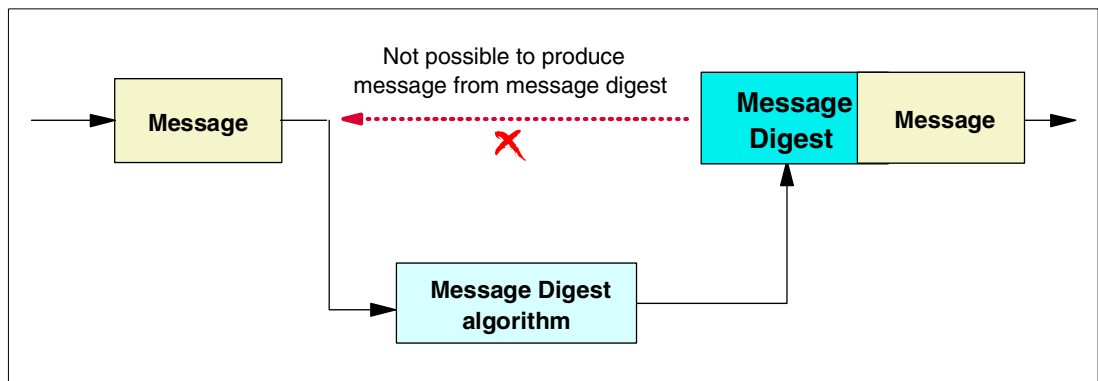


Figure 2-6 Message digest

The message digest should not be sent in the clear: Since the digest algorithms are well-known and no key is involved, a man-in-the-middle could not only forge the message but also replace the message digest with that of the forged message. This would make it impossible for the receiver to detect the forgery. The solution for this is to use a message digest algorithm that uses cryptography when creating the message digest — that is, to use a message authentication code, described in 2.4.2, “Message authentication codes (MAC)” on page 19.

Message digest algorithms

Common message digest algorithms are:

- | | |
|-------------------------|---|
| MD2 | Developed by Ron Rivest of RSA Data Security, Inc., this algorithm is mostly used for PEM (Privacy Enhanced Mail) certificates. MD2 is fully described in RFC 1319. Since weaknesses have been discovered in MD2, its use is discouraged. |
| MD5 | Developed in 1991 by Ron Rivest, the MD5 algorithm takes as input a message of arbitrary length and produces as output a 128-bit message digest of the input. The MD5 message digest algorithm is specified in RFC 1321, <i>The MD5 Message-Digest Algorithm</i> . Collisions have been found in MD5 (see Hans Dobbertin: <i>Cryptanalysis of MD5 Compress</i> , available at http://www.cs.ucsd.edu/users/bsy/dobbertin.ps). |
| SHA-1 | Developed by the National Security Agency (NSA) of the U.S. Government, this algorithm takes as input a message of arbitrary length and produces as output a 160-bit “hash” of the input. SHA-1 is fully described in standard FIPS PUB 180-1, also called the Secure Hash Standard (SHS). SHA-1 is generally recognized as the strongest and most secure message digesting algorithm. |
| SHA-256, SHA-512 | Developed by the National Security Agency (NSA) of the U.S. Government. The security of a hash algorithm against collision attacks is half the hash size and this value should correspond with the key size of encryption algorithms used in applications together with the message digest. Since SHA-1 only provides 80 bits of security against collision attacks, this is deemed inappropriate for the key lengths of up to 256 bits planned to be used with AES. Therefore, extensions to the Secure Hash Standard (SHS) have been developed. SHA-256 provides a hash size of 256 bits, while SHA-512 provides a hash size of 512 bits. |

2.4.2 Message authentication codes (MAC)

Figure 2-7 on page 20 shows a message authentication code (MAC) being created for a message. The first step is to use a hashing algorithm, such as MD5, to compute a message digest. That message digest is then encrypted with a key, and appended to the original message. Both the message and the associated MAC are then sent to the recipient. The assumption here is that the recipient shares the same key, so that he may recompute the message digest and encrypt it with the shared key. This result should match the MAC sent on the message.

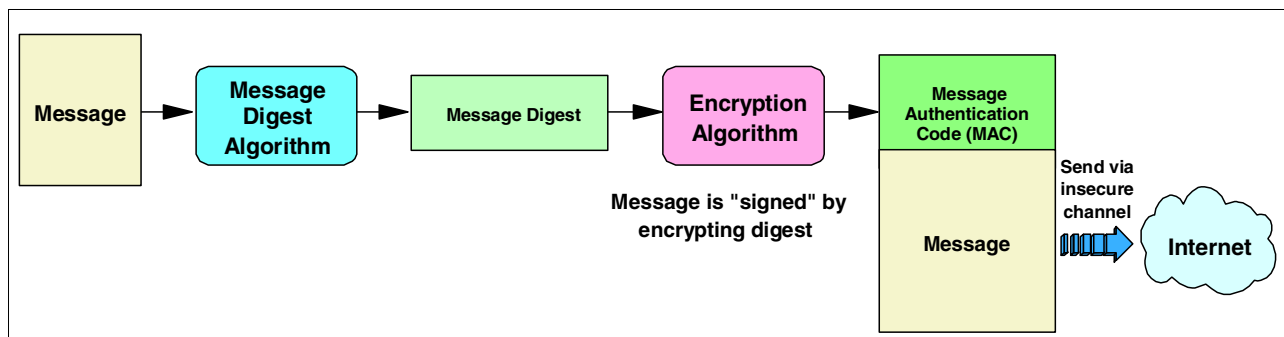


Figure 2-7 Message digest for data integrity

Secret-key cryptographic algorithms, such as DES, can be used for encryption with message digests. A disadvantage of using a secret-key algorithm is that since the receiver has the key that is used in MAC creation, this system does not offer a guarantee of non-repudiation. That is, it is theoretically possible for the receiver to forge a message and claim it was sent by the sender. Therefore, message authentication codes are usually based on public/private-key encryption in order to provide for non-repudiation. When a MAC is encrypted with a sender's private key, rather than a secret (symmetric) key, that MAC becomes a "Digital Signature". This is discussed further in 2.2.4, "Digital certificates" on page 13.

Keyed hashing for message authentication (HMAC)

H. Krawczyk and R. Canetti of IBM Research and M. Bellare of UCSD invented a method to create a message authentication code called HMAC, which is defined in RFC 2104 as a proposed Internet standard. A simplified description of how to create the HMAC is as follows. The key and the data are concatenated and a message digest is created. The key and this message digest are again concatenated for better security, and another message digest is created, which is the HMAC.

HMAC can be used with any cryptographic hash function. Typically, either MD5 or SHA-1 are used. In the case of MD5, a key length of 128 bits is used (the block length of the hash algorithm). With SHA-1, 160-bit keys are used. Using HMAC actually improves the security of the underlying hash algorithm. For instance, some collisions (different texts that result in the same message digest) have been found in MD5. However, they cannot be exploited with HMAC; therefore the weakness in MD5 does not affect the security of HMAC-MD5.

HMAC is now a PKCS#1 V.2 standard for RSA encryption (proposed by RSA Inc. after weaknesses were found in PKCS#1 applications). For further details, see <http://www.ietf.org/rfc.html>. HMAC is also used in the Transport Layer Security (TLS) protocol, the successor to SSL.

2.4.3 Digital signatures

Digital signatures are an additional means of securing data integrity. While data integrity only ensures that the data received is identical to the data sent, digital signatures go a step further: they provide non-repudiation. This means that the sender of a message (or the signer of a document) cannot deny authorship, similar to signatures on paper. As illustrated in Figure 2-8 on page 21, the creator of a message or electronic document that is to be signed uses a message digesting algorithm such as MD5 or SHA-1 to create a message digest from the data. The message digest and some information that identifies the sender are then encrypted with an asymmetric algorithm using the sender's private key. This encrypted information is sent together with the data.

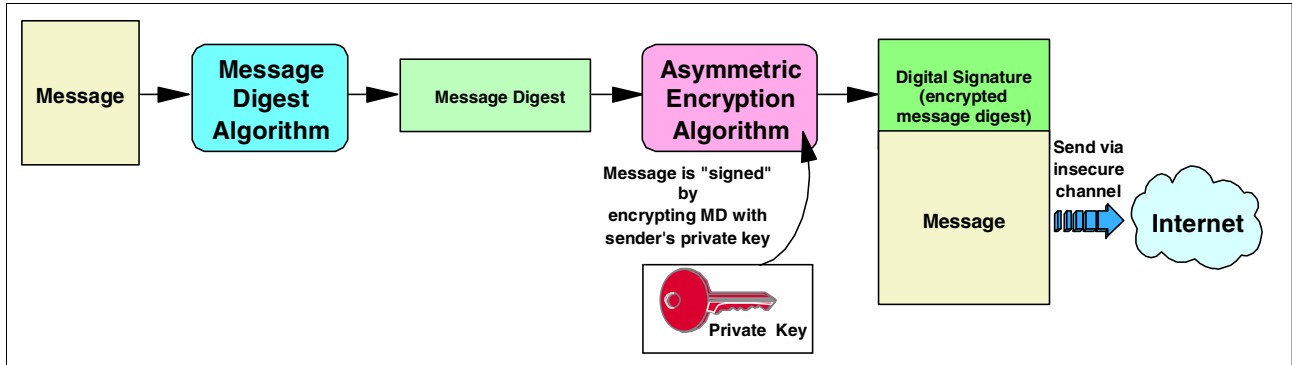


Figure 2-8 Digital signature creation

The receiver, as shown in Figure 2-9, uses the sender's public key to decrypt the message digest received. Then, he or she will use the message digesting algorithm to compute the message digest from the data received. If the computed message digest is identical to the one recovered after decrypting the digital signature, the signature is recognized as valid proof of the authenticity of the message.

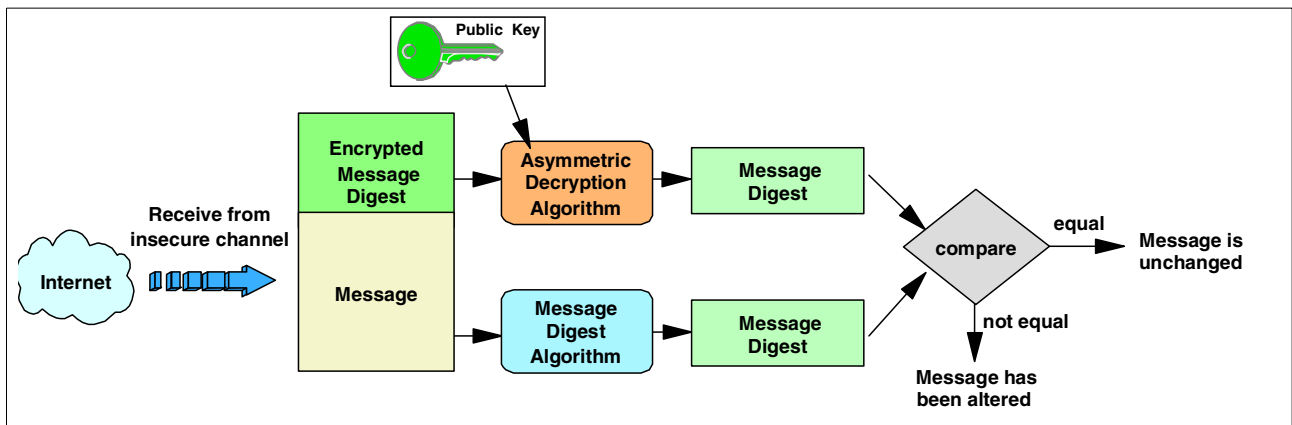


Figure 2-9 Digital signature verification

With digital signatures, only public-key cryptosystems can be used. If secret-key cryptosystems are used to encrypt the signature, it would be very difficult to make sure that the receiver (having the key to decrypt the signature) could not misuse this key to forge a signature of the sender. The private key of the sender is known to nobody else, so nobody is able to forge the sender's signature.

Note the difference between encryption using public-key cryptosystems and digital signatures:

- ▶ With encryption, the sender uses the receiver's public key to encrypt the data, and the receiver decrypts the data with his private key. This means everybody can send encrypted data to the receiver that only the receiver can decrypt. See Figure 2-10 on page 22 for a graphical representation.

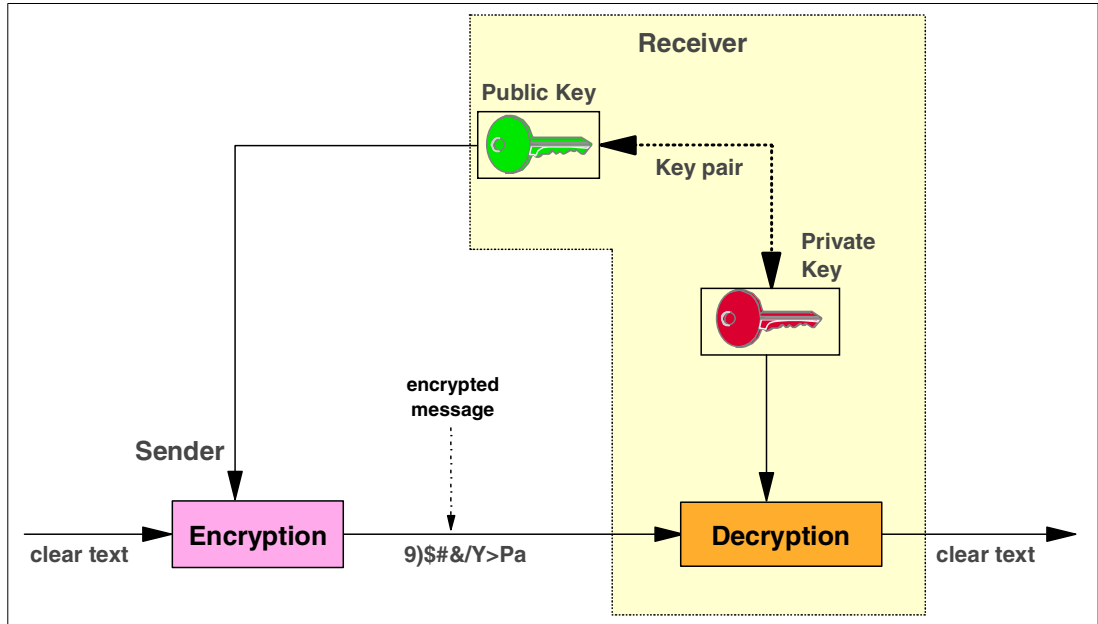


Figure 2-10 Encrypting data with the receiver's public key

- ▶ With digital signatures, the sender uses his private key to encrypt his signature, and the receiver decrypts the signature with the sender's public key. This means that only the sender can encrypt the signature, but everybody who receives the signature can decrypt and verify it.

The tricky part with digital signatures is the trustworthy distribution of public keys, since a genuine copy of the sender's public key is required by the receiver. A solution to this problem is provided by digital certificates, which was discussed in "Digital certificates" on page 13.

Securing z/OS with RACF

The chapters in this part address the ways to define the users of TCP/IP resources in a z/OS system. These resources are login accounts, Hierarchical File System (HFS) files, application sockets, IP ports, and other network resources.

A SAF product (we use RACF) is used to store information on users and their capabilities within the z/OS UNIX and non-UNIX environments.

For readers familiar with UNIX, we give an overview of RACF and of the differences between UNIX System Services security and traditional UNIX security. We then describe how UNIX System Services can be configured to provide the level of platform security required by your installation.



UNIX System Services security

This chapter discusses issues related to the security of z/OS UNIX System Services. For readers familiar with UNIX, we give an overview of RACF and of the differences between UNIX System Services security and traditional UNIX security. We then describe how UNIX System Services can be configured to provide the level of platform security required by your installation.

3.1 z/OS Security Server (RACF)

The standard access control system used on z/OS systems is the z/OS Security Server, particularly the Resource Access Control Facility (RACF) element of the z/OS Security Server. For convenience, we refer to the product as RACF throughout this redbook.

For a software access control mechanism to work effectively, it must be able to:

- ▶ Identify the person who is trying to gain access to the system
- ▶ Authenticate the user by verifying that the user is really that person
- ▶ Log and report attempts of unauthorized access to protected resources
- ▶ Control the access to resources
- ▶ Allow applications to use the RACF interfaces

3.1.1 Identification and authentication

RACF uses a user ID to identify the person who is trying to gain access to the system and a password to authenticate that identity. RACF uses the concept of only one person knowing a particular user ID and password combination to verify user identities and to ensure personal accountability. So, RACF determines:

- ▶ If the user is defined to RACF
- ▶ If the user has supplied a valid password, pass ticket, and a valid group name
- ▶ If the user's ID (UID) and group ID (GID) are valid on z/OS UNIX System Services (UID and GID are explained in 3.2.1, "Traditional UNIX security mechanisms" on page 29)

- ▶ If the user ID is in REVOKE status, which prevents a RACF-defined user from entering the system at all or entering the system with certain groups (if the user's group connection is revoked)
- ▶ If the user can use the system on this day of the week and at this time of day (an installation can impose restrictions)
- ▶ If the user is authorized to access the terminal (which can also include day and time restrictions for accessing that terminal)
- ▶ If the user is authorized to access the application

After authenticating the user's identity, RACF specifies the scope of the user's authorization for the current terminal session or batch job.

3.1.2 Alternatives to passwords

In a z/OS environment, RACF allows alternatives to passwords to be used for authenticating users, such as a pass ticket. In a z/OS UNIX System Services environment where users are also identified with numeric user identifiers (UIDs) and group identifiers (GIDs), these too may be used by RACF to control user access to system resources. Unlike user names or group names, these numeric IDs can be shared by more than one user or group, although sharing is not recommended.

In a client/server environment, RACF has the ability to map a client's digital certificate to a RACF user ID, the digital certificate being stored in the RACF database, or mapped by a certificate name filter rule. A digital certificate or digital ID, issued by a Certificate Authority, contains information that uniquely identifies the client.

The IBM HTTP Server for z/OS, for example, authenticates a client using the client's certificate over an SSL-secured session (the so-called SSL client authentication). The HTTP Server passes the client's digital certificate to z/OS UNIX System Services for validation. UNIX System Services passes the certificate to RACF to retrieve the RACF user ID from a mapping profile in the RACF database. This means that the RACF user ID and password of each client do not need to be supplied when accessing secure Web pages or other resources.

3.1.3 Checking authorization

After identifying and authenticating the user, RACF controls the interaction between the user and the resources in the system. RACF is called by resource managers to authorize a user's access to a resource. This includes the access level, for instance READ (for a user to read, display or copy a resource) or UPDATE (for a user to write, update or rewrite a resource). Access levels are hierarchical in RACF, so UPDATE includes READ and so on.

Figure 3-1 on page 27 shows how RACF uses the RACROUTE REQUEST=AUTH call to check authorization. The resource managers issue the other RACF calls in a similar manner.

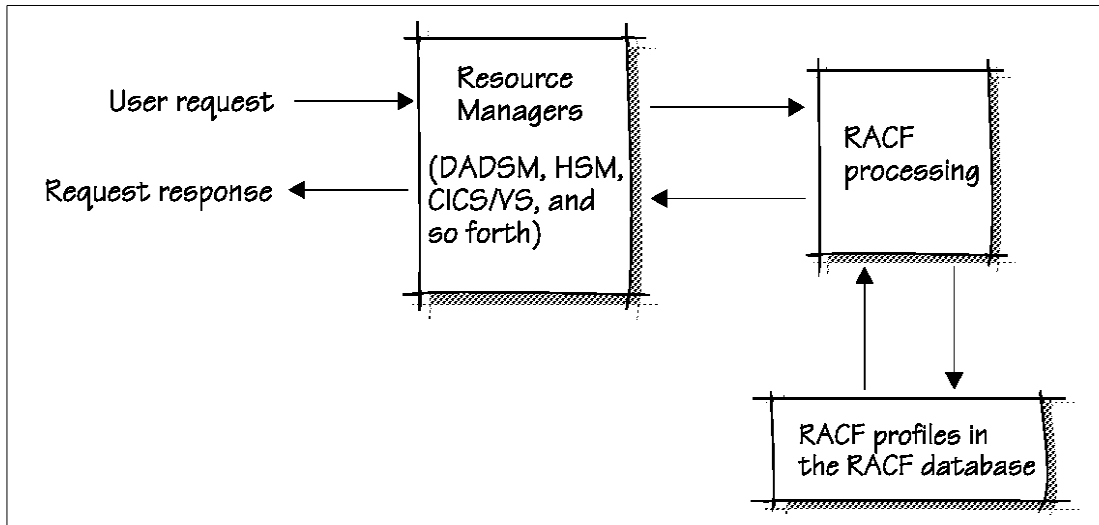


Figure 3-1 Example of RACROUTE REQUEST=AUTH processing

Before a user can access a resource, RACF does the following:

1. Checks the profiles to determine whether the user is authorized to access the resource. z/OS data sets are protected by profiles in the RACF class DATASET, but HFS files are protected differently. In the z/OS environment, an HFS itself is a z/OS data set protected by a DATASET profile that controls access to the data set. However, individual files and directories within the HFS are protected by means of permission bits as described in 3.3.4, “Access permission to HFS files and directories” on page 36.
2. Checks the security classification of the user and data.
3. Gives the user access to the resource if any of the following conditions is satisfied:
 - The resource is a data set and the high-level qualifier is the user’s user ID.
 - The user ID is in the access list with sufficient authority.
 - Any of the groups the user is connected to is in the access list with sufficient authority.
 - The universal access authority (UACC) or the access granted to ID(*) is sufficiently high.

3.1.4 Logging and reporting

RACF maintains statistical information, such as the date, time, and number of times that a user enters a system and the number of times a specific resource was accessed by any one user. Depending on the auditing options defined, RACF also writes security log records when it detects:

- ▶ Unauthorized attempts to enter the system
- ▶ Authorized or unauthorized attempts to access RACF-protected resources
- ▶ Authorized or unauthorized attempts to enter RACF commands

RACF writes all log records to the System Management Facility (SMF), a central recording facility in z/OS, as SMF type 80 records. SMF records all its data into data sets it uses in a round-robin fashion. Usually, installations have automated or semi-automated procedures to save and clear the SMF data sets. In the z/OS UNIX System Services environment, a large number of events are available that can be logged by RACF. Among them are events such as “check access to directory”, “check access to file”, “chmod”, “setuid”, “seteuid”, “mount file system”, and many more.

3.1.5 RACF and z/OS UNIX System Services

UNIX System Services security functions are implemented in RACF, partially as extensions to existing RACF functions and partially as new RACF functions. The security functions provided include user authentication, file access checking, and privileged user checking.

UNIX System Services users are defined with RACF commands. When a job starts or a user logs on, the user ID and password are verified by existing z/OS and RACF functions. When an address space requests a z/OS UNIX function for the first time, RACF does as follows:

- ▶ Verifies that the user is defined as a z/OS UNIX user.
- ▶ Verifies that the user's current connect group is defined as a z/OS UNIX group.
- ▶ Initializes the control blocks needed for subsequent security checks.

File access, including access to load modules for execution, is controlled based on the user's UNIX identity and the permissions associated with the files in question.

3.2 Security in UNIX systems

UNIX and z/OS UNIX System Security systems manage user identities differently. Table 3-1 contrasts some aspects of how user names and identities are handled by UNIX systems and z/OS UNIX.

Table 3-1 UNIX security comparison

Category	UNIX	z/OS MVS	z/OS UNIX
User identity	Users are assigned a unique UID: 4-byte integer and user name	Users are assigned a unique user ID: 1 to 8 characters	Users are assigned a unique user ID with an associated UID
Security identity	UID	user ID	UID for accessing traditional UNIX resources and the user ID for accessing traditional z/OS resources
Login ID	Name used to locate a UID	Same as the user ID	Same as the user ID
Special user	Multiple user IDs can be assigned a UID of 0	RACF administrator assigns necessary authority to users	Multiple user IDs can be assigned a UID of 0 or users can be permitted to BPX.SUPERUSER
Data set access	Superuser can access all files	All data sets controlled by RACF profiles	Superuser can access all HFS files; data sets controlled by RACF profiles

Category	UNIX	z/OS MVS	z/OS UNIX
Identity change from superuser to regular user	Superuser can use the su command to switch into any UID	APF-authorized program can invoke SAF service to change identity	<ol style="list-style-type: none"> 1. If FACILITY class profile BPX.DAEMON is not defined, the superuser can su into any other UID 2. If BPX.DAEMON is defined, the superuser must know the password of the other user ID or have access to SURROGAT class profile BPX.SRV.userid
Identity change from regular user to superuser	The su shell command allows change if user provides root's password	No provision for unauthorized user to change identity	The su shell command allows change if the user is permitted to the BPX.SUPERUSER FACILITY class profile or if the user provides the password of a user with a UID(0)
Identity change from regular user to another regular user	The su shell command allows change if user provides password	No provision for unauthorized user to change identity	The su shell command allows change if user provides password
Terminate user processes	Superuser can kill any process	MVS operator can cancel any address space	Superuser can kill any process, UNIXPRIV class profile can authorize non-UID(0) users
Multiple logins	Users can log in to a single user ID multiple times	Users can only log on to TSO/E once per user ID	Users can rlogin or Telnet multiple times to a single user ID in the UNIX shell, and log on once to TSO/E at the same time
Login daemons	inetd, rlogind, lm, and telnetd process user requests for login; a process is created with the user identity (UID)	TCAS and VTAM process user requests for logon; a TSO/E address space (process) is created with the user identity (user ID)	Users can log on to TSO/E or log in using one of the login daemons; in all cases, an address space is created with both an MVS identity (user ID) and a UID

3.2.1 Traditional UNIX security mechanisms

UNIX is not just one operating system. There are many different “flavors” of UNIX depending on the vendor, such as AIX by IBM, HP-UX by Hewlett-Packard, and Solaris by Sun. Also, some UNIX operating systems are released by not-for-profit groups such as Linux and FreeBSD. Therefore, we have to concentrate on the common characteristics of UNIX systems.

Security in traditional UNIX is largely implemented by means of the UID and GID and the use of permission bits. Additional security can be provided by means of access control lists (ACLs).

User ID and group ID

In the UNIX architecture, each user who has access to a system has a user name and an identification called a UID. The UID is a numeric value, typically in the range of 0 to $2^{(31)}-1$. Each UID can belong to one or more groups.

Each group is identified by its name and group ID (GID), a numeric value just like the UID. In some UNIX systems, a user can use only one group at a time, and can switch to another group, provided that the user is also listed as a member of this group. Other UNIX systems such as AIX allow a user to be connected to many groups at the same time, somewhat similar to the “list of groups” function in RACF.

Each object (file or directory) in the file system has nine permission bits. The nine permission bits are divided into three sets of three bits each:

- ▶ The first set of three bits shows the owning UID's permission.
- ▶ The next set of three bits shows the permission of the other users in the group that owns the file or directory.
- ▶ The last set of three bits shows the permission of anyone else with access to the file.

The three bits in each set indicate, respectively, read, write, and execute permission to the file. Read permission to a directory lets you list the files and subdirectories it contains. Write permission to a directory allows you to create, update, or delete an object in that directory. Execute permission to a directory lets you search a directory for a specified file.

To be able to access a file, not only the appropriate permission to the file but also permission to search the chain of directories from the root directory down to the file is required.

Access control lists (ACLs)

ACLs are a facility offered by some versions of UNIX, including AIX. The need for the functionality they provide is as follows.

The POSIX permission bits give a specific set of access permissions to only one group. However, there are cases when, for instance, one group needs read access to a file or directory while another group needs write access.

This is a problem that access control lists (ACLs) can solve. An ACL is a list of permissions attached to an object (file or directory). These permissions permit or deny access to the object. They can specify user IDs or group IDs, and allow or deny them separately read, write or execute rights. The degree of control and flexibility offered by ACLs allows system administrators to answer complicated user requirements where some users fulfill different roles and participate in different workgroups.

An object can have at most one ACL attached to it. If there is no ACL, its permission bits are used to determine its access attributes.

Conversely, the same ACL can be duplicated and applied to several files, to which it will grant identical permissions.

Auditing

Several versions of UNIX, including AIX, have full auditing. The system administrator can specify which objects (files or directories) are to be audited, and what kind of access should be audited. Audit log files are produced during the operation of the system.

Ideally, these logs should be reviewed periodically. Unfortunately, there is no standard way of filtering and querying audit log files under UNIX. For this task, most system operators rely on locally written scripts in AWK, Perl, or other text-processing languages.

3.3 z/OS UNIX System Services security

The security mechanisms in z/OS are integrated with the system. Each component that needs a service from the security product calls it with a standard interface called RACROUTE. The RACF component of the z/OS Security Server provides the needed services. z/OS UNIX System Services use the RACROUTE callable services to interface with RACF and receive the needed security services.

Note: Other security products are available, but they are not the focus of this redbook. This redbook exclusively refers to RACF, and statements made about z/OS UNIX System Services security take only RACF into account.

In z/OS, each user who may access the system has an identification called a user ID. For each user ID, a user profile is defined in class USER in the RACF database that contains all information about this user. The data elements for a user that are needed in the z/OS UNIX environment such as UID or home directory are defined in the *OMVS segment* of the user profile.

For organizational purposes and for ease of administration, users can be connected to groups. Groups are identified by a *group name* and are collections of users. A user who is connected to a group inherits the access rights that have been given to the group. Users can be connected to many groups and can make use of all of their groups' privileges at the same time. This property in RACF is called *list-of-groups checking*.

In UNIX systems, a superuser has the authority to access any files and to do all administration of users and resources. In a system like that, there is no superuser accountability and allowing a superuser to assume any other user identity does not add any security risk to the system.

In a z/OS system, there is a system of checks and balances between security administrators (users with the RACF attribute SPECIAL), auditors (users with the RACF attribute AUDITOR), and system programmers. In this type of system, allowing a superuser to switch into the identity of any z/OS UNIX user could be a security risk.

For this environment, additional security functions have been implemented in OS/390 UNIX System Services to ensure that overall system security is not negatively impacted by UNIX functions. These additional security functions are optional, and therefore z/OS UNIX System Services supports two levels of system security:

- ▶ UNIX level (the traditional method)
- ▶ z/OS UNIX level (with added security functions for z/OS)

Both levels are distinguished by the existence of at least one of the following two profiles in class FACILITY: BPX.DAEMON and BPX.SERVER.

3.3.1 UNIX level security

If the BPX.DAEMON and BPX.SERVER profiles in class FACILITY class are not defined, the system has UNIX-level security.

This level of security is for installations where superuser authority has been granted to system programmers and security administrators. These individuals already have permission to access critical data sets such as PARMLIB, PROCLIB, and LINKLIB, and they can exert total authority over the system.

Daemon programs run with superuser authority and can issue `setuid()`, `seteuid()`, and `__spawn()` to change the UID of any process to a different UID.

At the UNIX level of security, administrators need to be superusers and daemon programs such as `inetd` or `cron` must run with UID(0).

Note: We do not regard UNIX level security to be adequate for a z/OS UNIX system services environment and do not recommend running any system at this level of security. Consequently, the rest of this chapter assumes a system running with z/OS UNIX System Services security.

3.3.2 z/OS UNIX System Services level security

A system has z/OS UNIX System Services level security when at least one of the FACILITY class profiles BPX.DAEMON or BPX.SERVER is defined. At this level of security, additional security functions are active to increase the overall security of the system, especially better control of identity changes and improved superuser control.

This level of security is for customers with stricter security requirements who need to have some superusers maintaining the file system but want to have greater control over the z/OS resources that these users can access. Although BPX.DAEMON provides some additional control over the capabilities of a superuser, a superuser should still be regarded as a privileged user because of the large range of privileges the superuser is granted.

In a traditional UNIX environment, a user is associated with a UID. If the UID of a process is changed by a service such as `setuid()`, the identity of the user associated with the process also changes. This is not necessarily true in the z/OS UNIX system services environment, where a user is identified by a RACF user ID as well as a z/OS UNIX UID. When a program or user attempts to alter the user identity of a process, the result will vary depending on what authority is permitted. The following possibilities exist:

- ▶ The effective UID of a process is changed to UID(0) but the RACF user ID remains unchanged. This happens in the following cases:
 - When a shell user enters the `su` command without specifying a user ID in order to enter superuser mode.
 - When a TSO ISHELL user selects the **Enter Superuser Mode** menu option.
 - When a program such as SMP/E issues `seteuid()` to enter superuser mode.

Note: The RACF user ID associated with the process needs READ access permission to FACILITY class profile BPX.SUPERUSER in all cases above.

- ▶ A program or daemon first issues the `__passwd()` service to authenticate the RACF user ID to be switched to and then issues `setuid()` or `__spawn()` with target user ID to change the user ID and UID of the process or the child process, respectively. The caller of

__passwd() needs to supply the user ID and the password or PassTicket of the user. If the password is not supplied, the caller needs READ access to profile BPX.SRV.userid in class SURROGAT. Also, the address space of this process needs to be program-controlled (more on this later). Examples for this usage are:

- A user enters the `su` command with the `userid` parameter. The user must enter the password for the user ID or, with appropriate authority in class SURROGAT, the user can use the "-s" switch or press Enter at the password prompt.
- The FTP server, before changing the identity of the child process to the user ID and UID of the client, authenticates the user with user ID and password. For anonymous FTP, the FTP server needs access to profile BPX.SRV.anonymo in class SURROGAT, where user ID "anonymo" is the default RACF user ID for an anonymous FTP user.
- ▶ A daemon issues the `setuid()` or `spawn()` with user ID change service and just specifies the UID to be switched to without first authenticating the user. This is typically done by the cron daemon. The user ID of the process doing this needs to have *daemon authority* to be successful.
- ▶ A server program issues the service `pthread_security_np()` to create a thread running under this user ID and UID. The issuer of `pthread_security_np()` needs access to profile BPX.SERVER in class FACILITY. Also, the address space of this process needs to be program-controlled.

Note: BPX.DAEMON provides additional controls for the use of kernel services such as `setuid()`, which changes an issuer's UID in the z/OS UNIX environment. In z/OS UNIX, any user can issue a `setuid()` that follows a successful `__passwd()` call, which can be used to verify and/or change a user's password, to the target user ID. However, when a user want to change his/her user identity without knowing the target user's password, daemon authority is required.

As you might have noticed, the FTP server does not have to have access permission to BPX.DAEMON, because it can authenticate a target user with his password, which has been entered by the FTP client. For the anonymous FTP support, however, the server must have daemon authority, because no password is available for an anonymous user. We recommend you use the SURROGAT class profile rather than BPX.DAEMON to prevent possible security exposure.

The controlled program environment

In z/OS UNIX System Services, security critical functions, such as authenticating users or switching identities, require that the process runs in an address space that is a controlled program environment.

In a controlled program environment (also called a *program-controlled* address space), all programs that are loaded into the address space must either:

- ▶ Be loaded from an MVS program library (PDS or PDSE) that is defined in RACF class PROGRAM, or
- ▶ Be loaded from an HFS file that has the extended attribute "program controlled" (PROGCTL) turned on.

Any uncontrolled program from the HFS or an MVS library that is loaded into the address space will cause it to become uncontrolled, often called *dirty*. Once an address space becomes uncontrolled, it cannot be reverted into a controlled program environment. (There are fairly complicated methods in TSO/E involving the TSO/E Service Facility that can create a program-controlled task structure in an uncontrolled address space, but these are irrelevant in the z/OS UNIX environment.)

The requirement for a controlled program environment makes it more difficult for intruders and Trojan horse programs to misuse a security-critical process. All attempts to replace or modify code in the address space will cause it to become uncontrolled and the security-critical functions will fail.

It should be noted that the requirement to be in a library defined in class PROGRAM or to have the PROCTL extended attribute turned on also applies to DLLs and GWAPI plug-ins. A typical example for defining controlled libraries and programs is as follows:

```
RDEFINE PROGRAM * ADDMEM('SYS1.LINKLIB'//NOPADCHK) UACC(READ)
RALTER PROGRAM * ADDMEM('cee.SCEERUN'//NOPADCHK) UACC(READ)
RALTER PROGRAM * ADDMEM('imw.SIMWMOD1'//NOPADCHK) UACC(READ)
RALTER PROGRAM * ADDMEM('TCPIP.SEZALINK'//NOPADCHK) UACC(READ)
SETROPTS WHEN(PROGRAM) REFRESH
```

And an example of defining a module in the HFS a program-controlled is as follows:

```
BOCHE § SC57:/web/cert $ extattr +p pwapi.so
BOCHE § SC57:/web/cert $
```

Daemon authority

As mentioned in 3.3.2, “z/OS UNIX System Services level security” on page 32, switching into another UID and user ID without first authenticating the user requires the daemon authority. Why is this so potentially dangerous that a special authority from RACF is required? A program that can switch into the identity of any UID has almost the same authority as a superuser in traditional UNIX. This could make a z/OS system lose accountability, at least as far as user IDs that have a UID defined are concerned. When a process with daemon authority switches into another UID without authenticating the user first, RACF will search for a user ID that this UID belongs to in the same way it resolves UIDs to user IDs in an `1s -1` display. The process will then run with the UID and the user ID determined by RACF. As a consequence, a process with daemon authority can switch into any RACF user ID that has a UID defined.

It could be dangerous if a process with daemon authority were allowed to switch into UID(0) without further security control. If RACF would resolve UID(0) into a user ID such as OMVSKERN that has daemon authority by itself, this could allow someone to run a rogue program that assumes other user identities at will. To avoid this situation, the SUPERUSER parameter in member BPXPRMxx in SYS1.PARMLIB was created. It allows specifying a user ID with UID(0) that RACF will use when switching into UID(0) with daemon authority. This user ID (the default is BPXROOT) must not be defined with daemon authority.

For a process to have daemon authority, the following must all be true:

- ▶ The user ID associated with the process must be defined with an OMVS segment that specifies UID(0).
- ▶ The user ID associated with the process must have READ authority to profile BPX.DAEMON in class FACILITY.
- ▶ The process must run in a controlled program environment.

It can easily be seen that daemon authority is far-reaching and offers possibilities for compromising the security and integrity of the system. Therefore, READ access to profile BPX.DAEMON should be given to as few user IDs in the system as possible. Some examples of users/daemons that need READ access to BPX.DAEMON are:

- ▶ The UNIX System Services kernel user ID (the default is OMVSKERN)
- ▶ cron
- ▶ uucpd
- ▶ rlogind
- ▶ rshd

BPX.SERVER authority

As mentioned in 3.3.2, “z/OS UNIX System Services level security” on page 32, when a server program issues the service `pthread_security_np()` to create a thread running under this user ID and UID, its user ID needs access to profile BPX.SERVER in class FACILITY.

The access level to BPX.SERVER that the program needs is dependent on a number of conditions:

- ▶ Before issuing `pthread_security_np()`, the server program does not authenticate the user ID using `__passwd()`.
 - If the server’s user ID has UPDATE access to BPX.SERVER, the thread is created and all authorization requests for resources accessed by the thread are checked against the user ID or UID of the thread.
 - If the server’s user ID has READ access to BPX.SERVER, the thread is created and all authorization requests for resources accessed by the thread are checked against the user ID or UID of the thread as well as the user ID or UID of the server.
- ▶ Before issuing `pthread_security_np()`, the server program authenticates the user ID and specifies the correct password.
 - The server’s user ID needs READ access to BPX.SERVER; all authorization requests for resources accessed by the thread are checked against the user ID or UID of the thread.
- ▶ The user ID of the server program has READ access to profile BPX.SRV.userid in class SURROGAT
 - The server’s user ID needs READ access to BPX.SERVER. All authorization requests for resources accessed by the thread are checked against the user ID or UID of the thread.

In addition to the need to have access to BPX.SERVER, the process must also run in a controlled program environment to be able to use the `pthread_security_np()` service. One of the main users of this service is the IBM HTTP Server for z/OS.

3.3.3 Why is z/OS UNIX System Services a more secure UNIX?

z/OS UNIX System Services takes advantage of the inherent strengths of MVS and z/OS, including the security mechanisms of RACF. Working with RACF allows z/OS UNIX to provide these z/OS-exclusive enhancements to UNIX security:

1. No `/etc/passwd` file.

UNIX System Services relies on RACF for user authentication. This means that user information is not kept in `/etc/passwd` and `/etc/security/passwd`, respectively, and that all user administration is performed outside the z/OS UNIX System Services environment. It is easy to configure z/OS UNIX in a way that no access to the RACF database is possible from a UNIX process. This prevents brute-force password attacks against UNIX

passwords and stops intruders from altering user information from within z/OS UNIX.

2. Granularity of auditing and reporting.

UNIX System Services and RACF provide comprehensive auditing, allowing numerous events to be audited. The auditing information is written to SMF data sets, which can be made inaccessible even to superusers. Reporting is based on an open architecture that allows the use of practically any reporting package. This provides better detection of suspicious events.

3. Protection of daemon programs from modification and misuse.

Programs that perform security-critical functions must run in a controlled program environment. A modification to a module in such an address space or an attempt to load a module from another, uncontrolled library will cause the program to fail.

4. Superuser granularity control, a function that allows non-superusers authorized by RACF to use services that would otherwise require the user to be a superuser.

This can reduce the number of superusers required in a system.

5. The storage keys implemented in the S/390 and z900 hardware together with the concept of address in z/OS isolates applications from each other and from the operating system. A program that exceeds its allocated storage fails with a storage exception rather than overwriting storage areas of the operating system or other applications. This greatly reduces the possible damage that could be done by buffer overflows and similar problems.

6. TCP/IP stacks, ports, and network addresses can be protected with RACF to prevent unauthorized users and programs from using them.

7. The ability to assign different user identities (user IDs and UIDs) not only to processes but also to threads within a process allows access control in z/OS UNIX to be performed on a more granular basis than is possible in traditional UNIX environments. Specifically, programs such as the IBM HTTP Server for z/OS can run each thread under the client's identity. This allows the access authority of the client to be checked for all requests.

3.3.4 Access permission to HFS files and directories

To be able to access files in the HFS, a UID needs to be assigned to the process or thread that tries to open the file. This will be the case if the RACF user ID has a valid OMVS segment with an OMVS UID or if a default user with a valid UID has been defined.

When a process issues its first call to a z/OS UNIX service, it is said to be *dubbed*; it is given a z/OS UNIX identity that is based on the OMVS segments of the associated RACF user and group profiles.

The basic access algorithm can be deduced by reference to Figure 3-2 on page 37.

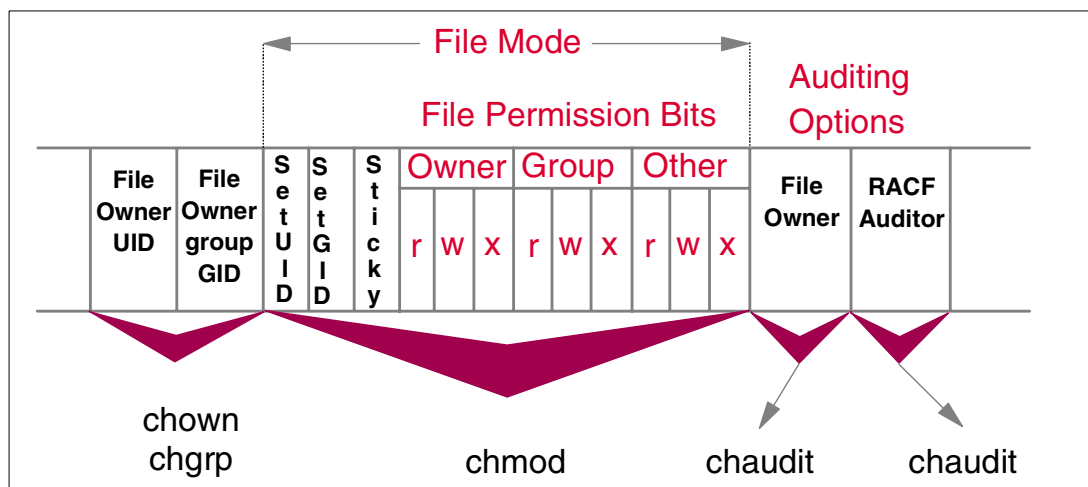


Figure 3-2 Hierarchical File System file security packet

Every file in the Hierarchical File System has a file security packet (FSP) assigned to it. This FSP contains the identification of the owning UID and GID along with information about what level of access (read, write, execute, or rwx for short) is granted to the three user categories that are considered to exist in this environment:

- ▶ The owner permissions

Any processes with an effective UID that matches the UID of the file owner can access the file under the defined permission. Please note that the owner of a file may restrict him or herself to read and execute (r-x) access. A superuser can change the file owner by issuing a **chown** shell command.
- ▶ The owning group permissions

Any processes with an effective GID that matches the GID of the file group can access the file under the defined permission. A superuser or the file owner can change the GID of the file by issuing a **chgrp** shell command.
- ▶ The other permissions

This permission class is applied to any process that is not the file owner, nor a member of the file owner's group. It is generally known as world access.

Note: A process running with an effective UID(0) can always read any file, even if not specifically permitted to do so by the permission bits.

Table 3-2 is an overview of types of access and the permissions granted by the accesses.

Table 3-2 File access types and permission bits

Access Type	Permission for File	Permission for Directory
Read	Permission to read or copy the contents	Permission to read, but not search, the contents
Write	Permission to change, add to, or delete from the contents	Permission to change, add, or delete directory entries, that is, HFS files, links, or subdirectories
Execute	Permission to run the file; this permission is used for executable files	Permission to search the directory

Notes:

- ▶ To access an HFS file, search permission to all directories in the path name of the file is required. Read permission is required for some options of some commands.
- ▶ To create new HFS files or directories, write permission to the directory in which they will be created is required.

To change the permission bits for a file, the file owner or superuser can use one of the following:

- ▶ The UNIX System Services ISPF shell
- ▶ The `chmod` shell command
- ▶ The `chmod()` API from a program

When accessing a file in the Hierarchical File System, the file system looks up the effective UID and GID of the current process and compares these to the owning UID and GID. If this fails, the check proceeds under the assumption of world access. Access checking is performed by the z/OS UNIX System Services kernel by using the appropriate callable service.

If RACF denies access, an error message will be logged to SYSLOG as shown in Figure 3-3.

```
ICH408I USER(BOCHE ) GROUP(SYS1 ) NAME(ULRICH BOCHE IBM GER)
/u/boche/.sh_history
CL(FSOBJ ) FID(01D7C4C7D6C5F2000320000000260000)
INSUFFICIENT AUTHORITY TO OPEN
ACCESS INTENT(RW-) ACCESS ALLOWED(GROUP ---)
```

Figure 3-3 Resource access authorization error

This message is the usual error message issued by RACF for access violations, but a short explanation of the different elements may be helpful:

- ▶ The first line identifies the user associated with the process or thread. USER is the RACF user ID, GROUP the current connect group, and NAME is the user name in the user profile.
- ▶ The second line identifies the failing resource, in this case the full path name of an HFS file.
- ▶ In the third line, CL identifies the RACF class, in this case FSOBJ. This is one of the z/OS UNIX classes that are only used to hold auditing options, so don't look for any profiles in this class. FSOBJ means *file system objects*, in other words, files. DIRACC would mean read/write access to a directory while DIRSRCH would appear if a directory search failed. For a description of all classes used, see Appendix A in *z/OS V1R3.0 Security Server RACF Security Administrator's Guide*, SA22-7683.
- ▶ . The FID parameter is normally only of interest to the IBM service organization for debugging purposes.
- ▶ The fourth line contains the reason for the failure.
- ▶ In the fifth and last line, ACCESS INTENT shows the access modes requested by the program. "RW-" in this case means the program tried to open the file for both reading and writing. ACCESS ALLOWED shows the access level that would have been allowed. GROUP means that one of the groups the user is connected to matched the owning group, so the group permissions were used, and they were "---", which means no access was allowed.

See Appendix C, “Default permissions for HFS files in z/OS UNIX” on page 443 for the default settings of the permission bits using different applications to create files or directories in the UNIX System Services.

3.3.5 Displaying files and directories

The shell command `ls` is used to list the contents of directories. The following example shows the output of the command `ls -l`:

```
-rw-r--r-- 1 OMVSKERN SYSPROG      9 Feb 28 14:58 syslog.pid
-rwxr-xr-x 1 OMVSKERN SYSPROG    157 Feb  4 1999 syslogd.start
-rwxr-xr-x 1 OMVSKERN SYSPROG    547 Feb 10 1999 t03dns.boot
-rw-r--r-- 1 OMVSKERN SYSPROG    201 Aug 10 1999 tcpipa.data
-rw-rw-rw- 1 OMVSKERN SYSPROG      0 Feb 11 1998 telnetd.stderr
-rw-r--r-- 1 OMVSKERN SYSPROG 20334 Feb 22 2000 testu03n
-rw-r--r-- 1 OMVSKERN SYSPROG     62 Jun 22 2000 trmd.r2615c.env
-rwxr-xr-x 1 OMVSKERN SYSPROG    119 Apr 27 1998 u.map
-rw-r--r-- 1 OMVSKERN SYSPROG    792 Mar  6 14:25 utmpx
-rw-r--r-- 1 OMVSKERN SYSPROG 15520 Feb  4 13:25 ylex.c
-rw-r--r-- 1 OMVSKERN SYSPROG 22559 Feb  4 13:25 yyparse.c
drwxr-xr-x 2 OMVSKERN SYSPROG    8192 Jan 29 1998 zoneinfo
ULRICH @ RA03:/etc>
```

The listing shows the user ID of the file owner and the group name of the owning group although the information in the HFS only contains the UID and GID, respectively. The cross reference between a UID and a user ID and a GID and a group name is done by a RACF service. Resolving a UID to a user ID (or a GID to group name) can be ambiguous if there is more than one user with the same UID assigned (or more than one group with the same GID).

Class UNIXMAP and VLF classes IRRUMAP and IRRGMAP

For the cross referencing, RACF uses profiles in class UNIXMAP. The profile names for UIDs are Unnnn, where nnnn is the UID number; for GIDs the profile names are Gnnnn. The user IDs associated with this UID are entered in the access list of the profile. If class UNIXMAP is used exclusively, RACF will always return the same user ID for a UID.

For performance reasons, VLF classes IRRUMAP and IRRGMAP are used. In the data space associated with IRRUMAP, UIDs and corresponding user IDs are stored at the time when this user ID is first used for a z/OS UNIX process after an IPL or after the table is rebuilt. The same is true for VLF class IRRGMAP and groups. When IRRUMAP and IRRGMAP are active, the user ID returned by RACF for a given UID will depend on the sequence in which user IDs have created processes. Therefore, the selection made by RACF among the user IDs sharing a common UID may appear rather arbitrary.

Application Identity Mapping (AIM)

In OS/390 V2R10, RACF introduces Application Identity Mapping (AIM) as a replacement for class UNIXMAP and VLF classes IRRUMAP and IRRGMAP. With AIM, RACF resolves UIDs to user IDs and GIDs to group names with the help of an alternate index into the RACF database. Making full use of this function requires all systems sharing a RACF database to be at the OS/390 V2R10 level or higher.

When AIM is used, RACF will always return the same user ID for a UID and the same group name for a GID, respectively.

3.3.6 UID/GID assignment to a process

As mentioned earlier, the first use of a z/OS UNIX service causes the z/OS address space to be *dubbed* into a z/OS UNIX process. The User Security Packet (USP) is an important control block created by dubbing. Among the information it holds are the UIDs relevant to the process:

- ▶ **Real UID:** At process creation, the real UID identifies the user who has created the process.
- ▶ **Effective UID:** Each process also has an effective UID. The effective UID is used to determine the owner access privileges of a process.
- ▶ Normally this value is the same as the real UID. It is possible, however, for a program that resides in the Hierarchical File System to have a special flag set that, when this program is executed, changes the effective UID of the process to the UID of the owner of the program. A program with this special flag set is said to be a set-user-ID program. This feature provides additional permissions to users while the set-user-ID program is being executed.
- ▶ **Saved UID:** Used to save the effective UID of a process. When a z/OS UNIX user tries to change the UID, the saved_UID field is checked to see if the UID is the original one.
- ▶ **Real GID:** At process creation, the real GID identifies the group of the user for which the process was created.
- ▶ **Effective GID:** Each process also has an effective group. The effective GID is used to determine the group access privileges of a process. Normally this value is the same as the real GID. Some programs, however, have a special flag set that, when the program is executed, changes the effective GID of the process to the GID of the owner of this program. A program with this special flag set is said to be a set-group-ID program. Like the set-user-ID feature, this provides additional permission to users while the set-group-ID program is being executed.
- ▶ **Saved GID:** Used to save the effective GID of a process. When a z/OS UNIX user tries to change the GID, the saved GID is checked to see if the GID is the original one.

The real UID and GID tell who really owns the process; the effective UID and GID are used for file access permission checks. The saved values of UID and GID are stored by the exec() function.

3.3.7 Defining UNIX System Services users

To define new UNIX System Services users, use the RACF command ADDUSER. The new user needs to have an OMVS segment with a UID. Also, the users default group needs to have a GID assigned. If needed, a new group can be defined using the ADDGROUP command, as follows:

```
ADDGROUP usrgrp OMVS(GID(10))
```

Then define the new user:

```
ADDUSER user01 DFLTGRP(usrgrp) OMVS(UID(20) HOME('/u/user01') -  
PROGRAM('/bin/sh'))
```

Note: For the UID, any value up to 2,147,483,647 can be used. However, UID(0) should only be assigned to superuser IDs. To avoid the task of manually keeping track of all the UIDs that are in use, we recommend using standard procedures to define users without OMVS segments, and using the TSO ISHELL to add the OMVS segment to the user ID. The TSO ISHELL will keep track of the UIDs it has issued and will assign the next available UID to a new user. Figure 3-4 on page 41 shows how to invoke the appropriate ISHELL function:

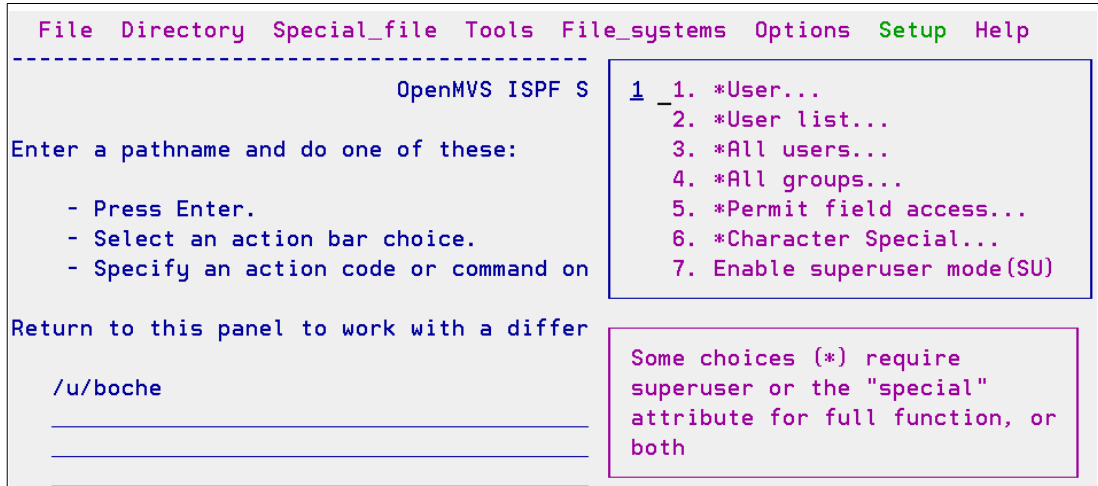


Figure 3-4 The setup drop-down menu in IDPF ISHELL

Selecting **Setup** -> **1. *User...** brings up the window shown in Figure 3-5 that will assign the next free UID to an existing user and also allows you to set the other OMVS segment parameters.

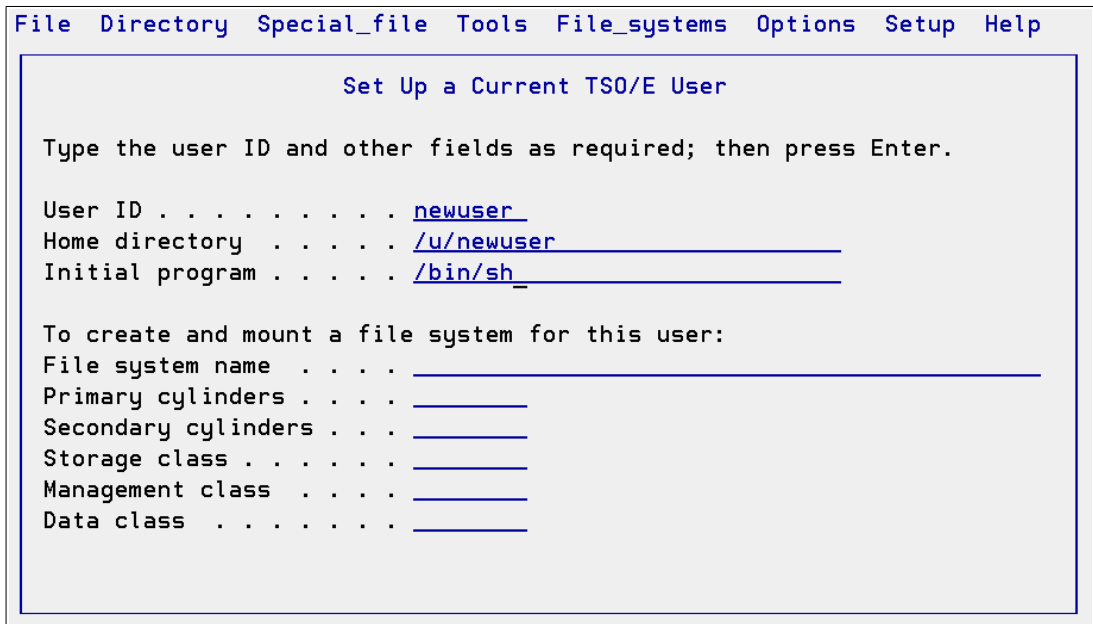


Figure 3-5 ISHELL window to assign a UID and other OMVS segment parameters to a user

For a user to be a UNIX System Services user, the user's default group must be a UNIX System Services group, that is, the group must be associated with a valid z/OS UNIX group ID.

For the new user01 to be able to log in to the UNIX System Services shell environment, you should create the home directory /u/user01 and give the user at least:

- ▶ Execute permission to the "/" and "/u" directories.
- ▶ Read, write and execute permission to the /u/user01 directory.

If you do not give user01 sufficient permission, any attempt to log in to the z/OS UNIX shell will fail. Note that defining /u/userid as the home directory for a user is just a frequently used convention; any valid directory name can be defined as a user's home directory.

3.3.8 Default user

The TCP/IP address space itself is an OS/390 UNIX application. As a consequence, any user of TCP/IP services needs to be a z/OS UNIX user. This means, for instance, that any FTP user who logs in with a RACF user ID needs to have a UID assigned or the FTP session will fail. The same is true for a user who points his or her Web browser to a URL that is protected and requires the user to authenticate with a user ID and password or a client certificate.

For various reasons, some installations do not want to assign individual UIDs to all these users. Some do not want all their users to be able to use the OMVS shell or the ISPF ISHELL; others do not want to spend the administrative overhead involved in defining OMVS segments for all users. These installations can define a default UID and GID for use with those user IDs that do not have an OMVS segment defined.

When you define an OMVS segment for the default user, you should consider the following entries:

► UID

You can use any value for default user and default group. Administrators often like to use a value for the default UID that makes it stand out from other, regular UNIX System Services users. Of course, assigning UID(0) to the default user would create a major security exposure.

► HOME

The home directory is the initial HFS directory for users who enter the TSO command OMVS or use Telnet to enter a z/OS UNIX shell. Generally, users of the default UID should not be shell users. Therefore, the following alternatives seem appropriate:

- Define a home directory where the default user does not have write permission, such as the root (/) directory.
- Define the HOME directory as the /tmp directory which is designed for temporary files created by different users.

► PROGRAM

The shell program is executed when a user enters the z/OS UNIX shell. The default shell program is /bin/sh. If you do not want users running in a shell environment with the default UID and GID, then define the PROGRAM parameter for the user's OMVS segment as:

```
PROGRAM('/bin/false')
```

This will cause any attempt to enter the shell to terminate. Note, however, that this does not restrict TSO users from using the ISHELL.

An example of the setup required for a default user and group follows:

1. Activate the FACILITY class and define the z/OS UNIX default user profile, BPX.DEFAULT.USER:

```
RDEFINE FACILITY BPX.DEFAULT.USER APPLDATA('UDSSUSR/USSGRP')
```

2. Define a default GROUP and USER:

```
ADDGROUP USSGRP OMVS(GID(17))  
ADDUSER USSUSR DFLTGRP(USSGRP) NAME('USS DEFAULT USER') -  
OMVS(UID(88) HOME(/) PROGRAM('/bin/false'))
```

Note: You can use any value for UID and GID, except zero for the UID.

3. Refresh the FACILITY class profile:

```
SETRPTS RACLIST(FACILITY) REFRESH
```

The format of the application data is exactly as shown when a default is being set up for both USER and GROUP OMVS segments. To set up a default for the USER OMVS segment only, the format is:

```
APPLDATA('USSUSR')
```

There is no option to set up a default GROUP OMVS segment without a user.

Note: Because many z/OS UNIX processes now run with the default user's UID, the number of allowed processes per UID as defined in BPXPRMxx with the MAXPROCUSER statement may be too low.

Profile BPX.DEFAULT.USER in class FACILITY is used as follows:

- a. A user requests a UNIX service and the kernel dubs the user.
- b. The kernel calls the security product to extract the UID, GID, HOME, and PROGRAM information.
- c. The security product attempts to extract the OMVS segment associated with the user. If it is not defined, it tries to extract and use the OMVS segment for the default user that was listed in the BPX.DEFAULT.USER profile.

Note: If you allow users to access UNIX System Services resources using the default UID and GID, be aware that these users can access all “world-readable” HFS files and directories. In an HFS environment, it is very often infeasible to avoid world-readable files because of the limited possibilities offered by the POSIX permission bits. Therefore, the use of the default UID and GID requires careful planning and risk assessment.

Also, if you have trusted or privileged started tasks defined in the STARTED class or in ICHRIN03, these started tasks will be superusers if they use a z/OS UNIX service even if they do not have an OMVS segment defined.

3.3.9 Superuser

In any UNIX system, a user with a UID of zero has special privileges. This user is said to be a superuser. Superusers are very powerful users, so the number of users with UID(0) should be kept to a minimum and these IDs should be handled with great care.

At installation, you have to define at least one superuser and a group to which the superuser will belong. The group could be an existing group as long as it has a GID assigned.

```
ADDGROUP OMVSGRP OMVS(GID(1))
```

Note: You can use any group ID for the group to which the superuser will belong.

Then define the superuser:

```
ADDUSER OMVSKERN DFLTGRP(OMVSGRP) OMVS(UID(0) HOME('/') PROGRAM('/bin/sh'))
```

In this case, the superuser is named OMVSKERN and its default group is OMVSGRP.

In most UNIX systems, a UID of zero gives unlimited rights to access and manipulate files in the HFS and to change the identity of the process that is running with a UID of zero. In most UNIX systems, this means that the system administrator normally has a UID of zero, and any server programs that need to change the identity of forked processes run with a UID of zero.

In a z/OS UNIX system running with z/OS UNIX System Services security, limitations to the authorities of a superuser apply, such as:

- ▶ Using a service such as `setuid()`, even a superuser can only switch into another UID without specifying the password if the user ID has READ access to profile BPX.DAEMON in class FACILITY.
- ▶ The use of the `su` command to switch into another user ID requires the user ID's password or READ access to profile BPX.SRV.userid in class SURROGAT.
- ▶ Extended attributes for HFS files such as program-controlled (PROGCTL), APF-authorized (APF), and shared library (SHARELIB) can only be set with READ authority to the appropriate RACF profile in class FACILITY (BPX.FILEATTR.PROGCTL, BPX.FILEATTR.APF, or BPX.FILEATTR.SHARELIB, respectively).
- ▶ Use of the `ptrace` function of `dbx` to debug programs running with APF authority or with BPX.SERVER authority requires READ access to profile BPX.DEBUG in class FACILITY.

Superuser granularity

Even with the restrictions imposed on superusers in z/OS UNIX System Services, a superuser still has far-reaching authority. Therefore, it should be each installation's goal to keep the number of users with superuser authority to an absolute minimum. This is an important contribution to the overall security and accountability of the z/OS system.

A number of system programmers and administrators do not need full superuser authority. They just need to perform a few selected functions that cannot normally be executed without superuser authority. In this situation, the number of superusers (or users able to switch into superuser mode through access to BPX.SUPERUSER) that are needed in the system can be reduced by using a function called *superuser granularity*.

Superuser granularity allows a non-superuser to successfully execute a function that would normally require superuser authority if the user has access to a certain resource in RACF class UNIXPRIV. For example, a file system can normally only be mounted by a superuser. However, a user with a non-UID(0) user ID can successfully mount file systems if the user ID has access to profile SUPERUSER.FILESYS.MOUNT. READ access will allow file systems with the `nosetuid` option, while UPDATE access will also allow the user to mount file systems with the `setuid` option.

Table 3-3 shows the resource names that are used in class UNIXPRIV and lists the privileges associated with each resource.

Table 3-3 Resource names in the UNIXPRIV class for z/OS UNIX privileges

Resource name	z/OS UNIX privilege	Access required
CHOWN.UNRESTRICTED	Allows all users to use the chown command to transfer ownership of their own files.	NONE
SUPERUSER.FILESYS.CHOWN	Allows users to use the chown command to change ownership of any file.	READ

Resource name	z/OS UNIX privilege	Access required
SUPERUSER.FILESYS ¹	Allows users to read any HFS file and to read or search any HFS directory.	READ
	Allows users to write to any HFS file and includes privileges of READ access.	UPDATE
	Allows users to write to any HFS directory and includes privileges of UPDATE access.	CONTROL (or higher)
SUPERUSER.FILESYS.MOUNT	Allows users to issue the mount command with the nosetuid option and to unmount a file system mounted with the nosetuid option.	READ
	Allows users to issue the mount command with the setuid option and to unmount a file system mounted with the setuid option.	UPDATE
SUPERUSER.FILESYS.QUIESCE	Allows users to issue the quiesce and unquiesce commands for a file system mounted with the nosetuid option.	READ
	Allows users to issue the quiesce and unquiesce commands for a file system mounted with the setuid option.	UPDATE
SUPERUSER.FILESYS.PFSCTL	Allows users to use the pfscctl() callable service.	READ
SUPERUSER.FILESYS.VREGISTER ²	Allows a server to use the vreg() callable service to register as a VFS file server.	READ
SUPERUSER.IPC.RMID	Allows users to issue the ipcrm command to release IPC resources.	READ
SUPERUSER.PROCESS.GETPSENT	Allows users to use the w_getpsent callable service to receive data for any process.	READ
SUPERUSER.PROCESS.KILL	Allows users to use the kill() callable service to send signals to any process.	READ
SUPERUSER.PROCESS.PTRACE ³	Allows users to use the ptrace() function through the dbx debugger to trace any process. Allows users of the ps command to output information on all processes. This is the default behavior of ps on most UNIX platforms.	READ
SUPERUSER.SETPRIORITY	Allows users to increase own priority.	READ
<p>Notes:</p> <ol style="list-style-type: none"> 1. Authorization to the SUPERUSER.FILESYS resource provides privileges to access only local Hierarchical File System (HFS) files. No authorization to access Network File System (NFS) files is provided by access to this resource. 2. The SUPERUSER.FILESYS.VREGISTER resource authorizes only servers, such as NFS servers, to register as file servers. Users who connect as clients through file server systems, such as NFS, are not authorized through this resource. 3. Authorization to the resource BPX.DEBUG in the FACILITY class is also required to trace processes that run with APF authority or BPX.SERVER authority. For more information about administering BPX profiles, see <i>z/OS V1R2.0 UNIX System Services Planning</i>, GA22-7800. 		

SETROPTS RACLIST needs to be issued for class UNIXPRIV to make the profiles resident in storage. Therefore any addition or modification to profiles in this class requires the profiles to be refreshed using the command:

```
SETROPTS RACLIST(UNIXPRIV) REFRESH
```

3.3.10 Started task user IDs

The association of a user ID with a started procedure can be done with profiles in resource class STARTED. This allows for a dynamic definition of started procedures and their associated user IDs as opposed to the use of the ICHRIN03 table which required an IPL of the system.

Typical entries in this resource class originating from UNIX System Services are the procedures for TCP/IP, FTPD, InetD, syslogd, the HTTP Server and a number of other applications.

The following example shows how to define a resource profile in class STARTED for the z/OS HTTP Server:

```
SETROPTS GENERIC(STARTED)
SETROPTS CLASSACT(STARTED) RACLIST(STARTED)
```

The above two commands are needed only if you have not configured the RACF STARTED class in your z/OS system. The next command (RDEFINE) associates the user WEBSRV in group USSGRP with the started task WEBSERV:

```
RDEFINE STARTED WEBSERV.* STDATA(USER(WEBSRV) GROUP(USSGRP) -
PRIVILEGED(NO) TRUSTED(NO) TRACE(NO))
SETROPTS RACLIST(STARTED) REFRESH
```

The profiles of RACLISTed classes are cached in a data space. Therefore, you need to refresh class STARTED every time you have made changes to any profiles.

To display the definition for a particular started class profile, use the RACF GENERAL RESOURCE SERVICES - DISPLAY panel or issue the following command:

```
RLIST STARTED WEBSERV.* STDATA
```

Figure 3-6 is a sample of the output produced by the RLIST command.

CLASS	NAME				
STARTED	WEBSERV.* (G)				
LEVEL	OWNER	UNIVERSAL ACCESS	YOUR ACCESS	WARNING	
00	SYS1	NONE	NONE	NO	
.....					
N					
STDATA INFORMATION					

USER= WEBSRV					
GROUP= USSGRP					
TRUSTED= NO					
PRIVILEGED= NO					
TRACE= NO					

Figure 3-6 RLIST output for STARTED class profile

When you start the Web server with the operator command START WEBSRV, you will see the following messages on the z/OS console:

```
IEF695I START WEBSERV WITH JOBNAME WEBSERV IS ASSIGNED TO  
USER WEBSERV, GROUP OMVSRP
```

3.3.11 FACILITY class profile BPX.SUPERUSER

This profile can be used to allow users with a nonzero UID to change the effective UID of their process into a UID of zero. If you have users who occasionally, as part of their daily system work, need to have superuser privileges, you can assign them a nonzero UID and permit them use of BPX.SUPERUSER. The alternative is to define all such users with a permanent UID of zero, which means that they perform all their work as superusers, even work that does not require them to be superusers. By using this profile you can limit the amount of time any user runs in superuser mode. Remember that in superuser mode, the user can, perhaps by mistake, delete the root file system of your Hierarchical File System. If a user who is permitted access to BPX.SUPERUSER logs on to the z/OS UNIX interactive shell, the user can type in a command to switch into superuser mode, and an exit command to return to his or her normal nonzero UID environment.

Below are examples of definitions for users who can switch to superuser status.

Before defining these users, you need to create profile BPX.SUPERUSER in class FACILITY. We assume the FACILITY class is already defined and activated. Enter the RDEFINE command shown in the following example to define this profile:

```
RDEFINE FACILITY BPX.SUPERUSER UACC(NONE)
```

Permit all users who need superuser authority to this profile. We give the TSO/E user ID SYSPROG permission to use **su** to obtain superuser authority. It is assumed that the default group for SYSPROG is set up with a GID.

Use the RACF commands shown in the following example:

```
ALTUSER SYSPROG OMVS(UID(7) HOME('/u/sysprog') PROGRAM('/bin/sh'))  
PERMIT BPX.SUPERUSER CLASS(FACILITY) ID(SYSPROG) ACCESS(READ)
```

Another alternative is to define a group, say SUPERUSR. You would then add users that need superuser permission to the group. To add the user ID SYSPROG to this group and then permit this group to the BPX.SUPERUSER profile, enter:

```
CONNECT (SYSPROG) AUTH(USE) GROUP(SUPERUSR) OWNER(SYS1)  
PERMIT BPX.SUPERUSER CLASS(FACILITY) ID(SUPERUSR) ACCESS(READ)
```

A user who does not have READ permission to the FACILITY class profile BPX.SUPERUSER and wants to change into superuser status will receive the following error message:

```
OZECH @ RA03:/u/ozech>su  
FSUM5011 su: User not authorized to obtain superuser authority
```

A daemon program that wants to make use of the authority to BPX.SUPERUSER to gain superuser status needs to invoke the seteuid() service. Therefore, using a non-UID(0) user ID with access to BPX.SUPERUSER instead of a superuser user ID for a daemon or server program is a solution only if the program does indeed support this function.

3.3.12 FACILITY class profile BPX.DAEMON

The profile BPX.DAEMON is very important to the concept of z/OS UNIX level security and the overall system security in z/OS. We discuss this concept extensively in 3.3.2, “z/OS UNIX System Services level security” on page 32. If you are not familiar with identity switching, daemon authority, and the controlled program environment, please go to this section.

To enable z/OS UNIX level security, define profile BPX.DAEMON in class FACILITY:

```
RDEFINE FACILITY BPX.DAEMON UACC(NONE)
```

Note: Activating z/OS UNIX level security in a system that did not use this level of security before could cause some daemon programs to fail. Before defining this profile or the FACILITY class profile BPX.SERVER, you should make sure that all daemons that use security-critical services are loaded from a controlled program environment and that appropriate authority to SURROGAT class profiles has been established. For details see “The controlled program environment” on page 33.

If this is the first FACILITY class profile that your installation is going to use, you need to activate the FACILITY class with the following commands:

```
SETRPTS CLASSACT(FACILITY) GENERIC(FACILITY) AUDIT(FACILITY)
SETRPTS RACLIST(FACILITY)
```

Depending on the technique you use to start your server programs, some of them may inherit their initial user identity from the kernel address space. This will be the case if you start server programs from the /etc/rc shell script that is executed during startup of UNIX System Services. In that case, the user ID of the kernel address space must be authorized to the BPX.DAEMON resource:

```
PERMIT BPX.DAEMON CLASS(FACILITY) ID(OMVSKERN) ACCESS(READ)
```

If you start server programs via z/OS start commands or from shell scripts that execute after startup of z/OS UNIX, you may need to give some of these user IDs the access to the BPX.DAEMON FACILITY class resource.

User ID BPXROOT

In “Daemon authority” on page 34, we explain the use of user ID BPXROOT. This user ID should not be used to log on to any interactive services, so it is appropriate to define it as a protected user ID (using the NOPASSWORD parameter). To define the BPXROOT user ID, enter:

```
ADDUSER BPXROOT DFLTGRP(OMVSGRP) OMVS(UID(0) HOME('/') -
PROGRAM('/bin/sh')) NOPASSWORD
```

In SYS1.PARMLIB member BPXPRMxx, the SUPERUSER parameter can be used to specify a user ID to be used instead of BPXROOT.

Note: Make sure you do not give the BPXROOT user ID or its replacement READ (or higher) access to profile BPX.DAEMON in class FACILITY. As we explain in “Daemon authority” on page 34, this would be a security exposure.

3.3.13 Additional BPX.* FACILITY class profiles

For security reasons, you may need to define these additional FACILITY class profiles:

▶ **BPX.DEBUG**

Users with read access to profile BPX.DEBUG in class FACILITY can use the ptrace function of dbx to debug programs that run APF authorized or authority to profile BPX.SERVER in class FACILITY.

▶ **BPX.FILEATTR.APF**

This controls which users are allowed to set the APF authorized attribute for an HFS file. This authority allows the user to create a program that will run APF authorized. This is similar to the authority of allowing a programmer to update SYS1.LINKLIB or SYS1.LPALIB.

▶ **BPX.FILEATTR.PROGCTL**

This controls which users are allowed to set the program-controlled attribute for an HFS file. Programs marked with this attribute can execute in program-controlled address spaces.

▶ **BPX.FILEATTR.SHARELIB**

OS/390 V2R9 introduced the concept of user and system-shared library objects. Shared libraries allow multiple processes to share subroutines in object libraries more efficiently.

Programs in a user-shared library (filenames with a suffix of “.so”) are loaded into the kernel-shared library region data space on a page boundary. Programs in a system-shared library are loaded into the kernel-shared library region data space on a megabyte boundary, use a single-page table for all address spaces that use them (similar to the LPA) and do not interfere with the virtual storage used by the application program.

A program in the HFS that has the shared library extended attribute set is loaded as a system-shared library program. To prevent misuse of system-shared libraries, profile BPX.FILEATTR.SHARELIB in class FACILITY controls who can set this attribute.

These are sample RACF definitions for all three BPX.FILEATTR profiles in class FACILITY:

```
RDEFINE FACILITY BPX.FILEATTR.APF UACC(NONE)
PERMIT BPX.FILEATTR.APF CLASS(FACILITY) USER(userid) ACCESS(UPDATE)
RDEFINE FACILITY BPX.FILEATTR.PROGCTL UACC(NONE)
PERMIT BPX.FILEATTR.PROGCTL CLASS(FACILITY) USER(userid) ACCESS(UPDATE)
RDEFINE FACILITY BPX.FILEATTR.SHARELIB UACC(NONE)
PERMIT BPX.FILEATTR.PROGCTL CLASS(FACILITY) USER(userid) ACCESS(UPDATE)
SETROPTS CLASS(FACILITY) REFRESH
```

▶ **BPX.SERVER**

This FACILITY class profile is explained in detail in “BPX.SERVER authority” on page 35.

▶ **BPX.SMF**

In the non-UNIX z/OS environment, a program needs to be APF authorized or in supervisor state or system key to be able to cut an SMF record. This protects the System Management Facilities from misuse by rogue programs. Such programs might purposefully create large amounts of SMF records to fill up all SMF recording data sets and either attempt a denial-of-service attack on the system (if an SMF full conditions halts the system) or to cover up illicit activity that might otherwise have been logged to SMF.

In z/OS UNIX, it is not appropriate to require daemons that need to cut SMF records to be APF authorized. To minimize the exposure that rogue UNIX programs might attack System Management Facilities, READ authority to FACILITY class profile BPX.SMF is required for z/OS UNIX applications to be able to cut SMF records.

► **BPX.STOR.SWAP**

This controls which users can make address spaces nonswappable. Users permitted with at least READ access to BPX.STOR.SWAP can invoke the `__mlockall()` function to make their address spaces either nonswappable or swappable.

When an application makes an address space nonswappable, it may cause additional real storage in the system to be converted to preferred storage.

Because preferred storage cannot be configured offline, using this service can reduce the installation's ability to reconfigure storage in the future. Any application using this service should warn the customer about this side effect in its installation documentation.

► **BPX.WLMSEVER**

This controls access to the WLM server functions `_server_init()` and `_server_pwu()`. It also controls access to these C language WLM interfaces:

- `QuerySchEnv()`
- `CheckSchEnv()`
- `DisconnectServer()`
- `DeleteWorkUnit()`
- `JoinWorkUnit()`
- `LeaveWorkUnit()`
- `ConnectWorkMgr()`
- `CreateWorkUnit()`
- `ContinueWorkUnit()`

A server application with read permission to this FACILITY class profile can use the server functions, as well as the WLM C language functions, to create and manage work requests.

3.3.14 Programs in the Hierarchical File System

The implementation of UNIX services in z/OS introduces a new location for executable programs: the Hierarchical File System. In UNIX terms, a program is an executable file, or for short, an executable. In MVS, the non-UNIX z/OS environment, we work with load modules that reside in load libraries.

A search for an MVS load module is performed in a predefined sequence of steps that include already loaded modules, STEPLIB or JOBLIB libraries, LPA resident modules, and modules that reside in libraries that are included in the system link list as specified in the LNKSTxx member of SYS1.PARMLIB. There are techniques in standard MVS to work with APF authorized load modules, which are programs that use certain authorized functions in an MVS system, for example, changing the MVS identity of an address space or a task in an address space with the RACROUTE REQUEST=VERIFY macro.

Since the HFS is a UNIX-style file system, z/OS UNIX uses UNIX-style techniques when searching for executable files in the HFS. The order of search is specified via an environment variable called PATH. Because every user may set his or her own PATH variable, it is relatively simple for a user to specify that the user's own programs should execute instead of system programs or commands. The user can, for example, assign the following value to the PATH environment variable:

```
./bin:/u/hdm:/usr/lpp/internet/www/Admin
```

The leading dot (.) specifies that the search for an executable file should begin in the current directory.

If the executable file is not found in the current directory, then the search continues through the `/bin`, `/u/hdm`, and finally `/usr/lpp/internet/www/Admin` directories.

Note: A superuser or a user with authority to BPX.SUPERUSER who has entered superuser mode should not use a PATH that contains the current directory. If it is absolutely necessary to use the current directory, it should be placed at the end of the search path.

As you can see from the above example, it is quite simple for a user to execute a different program with the same name in place of a system command or system program.

Processes that use security-critical functions need to run in a controlled program environment (for details see “The controlled program environment” on page 33) to make sure that all programs in the address space are loaded from a trusted source.

The sticky bit

The sticky bit was invented with early versions of UNIX where the process of loading an executable file into memory was very resource consuming. By setting the sticky bit in the file system for the executable file, the executable file was kept in storage even after it had finished executing, ready to be used by other users in the UNIX system. It was kept in storage until the system was shut down. Nowadays systems are not as constrained in storage as they used to be, so the original meaning of the sticky bit has vanished and it is now often used for other purposes.

UNIX System Services uses the sticky bit to bypass loading of an executable from the Hierarchical File System. If an executable file has the sticky bit turned on as shown in Figure 3-7, the executable file in the HFS will not be used, but z/OS will instead turn to the standard MVS search order to look for a copy of the executable file in an MVS load library.

```
/usr/lpp/internet: >ls -l
total 40
-rw-r--r-- 2 WEBADM IMWEB envvars
-rw-r--r-- 2 WEBADM IMWEB httpd_msg.cat
drwxr-xr-x 2 WEBADM IMWEB IBM
-rwxr--r-T 2 WEBADM IMWEB IMWCGIBN
Drwxr-xr-x 2 WEBADM IMWEB logs
Drwxr-xr-x 3 WEBADM IMWEB Samples
Drwxr-xr-x 10 WEBADM IMWEB ServerRoot
/usr/lpp/internet: >
```

Figure 3-7 Sticky bit

Above is an example of a sticky bit setting for an executable file. The T in the file security packet for the file called IMWCGIBN indicates that the sticky bit has been set.

When the z/OS UNIX program loader finds an executable file with the sticky bit on, it does not load the program from the Hierarchical File System, but passes the load request on to the MVS contents supervisor, which then searches for the requested module in one of the following libraries:

- ▶ STEPLIB, if it has been allocated to the current process
- ▶ The LPA
- ▶ Link libraries as they are defined in the LNKLSTxx or PROGxx members of SYS1.PARMLIB

If a process needs to run in a controlled program environment, all STEPLIB and LNKSTxx libraries where modules for this address space are loaded from have to be defined in class PROGRAM. The LPA is always considered a controlled environment, so LPA libraries do not need to be defined in class PROGRAM.

Extended attributes

Some files in the Hierarchical File System need special permissions over and above those normally used for UNIX files, such as attributes that:

- ▶ Mark an executable file as program controlled
- ▶ Mark an executable file as APF authorized
- ▶ Allow a program to be loaded into a shared address space
- ▶ Mark a program as a system-shared library object

In the Hierarchical File System, *extended attributes* can be set to define these special permissions with the **extattr** shell command:

- ▶ **extattr +p** marks an executable file as program controlled. See BPX.FILEATTR.PROGCTL on page 49 for the authority needed to set this attribute.
- ▶ **extattr +a** marks an executable file as APF authorized. Note that the program needs to be linked with AC=1 if it is to be invoked as a job step program. See BPX.FILEATTR.APF on page 49 for the authority needed to set this attribute.
- ▶ **extattr +l** marks an executable file as a system-shared library object. The program will be loaded into the kernel-shared library region data space. See BPX.FILEATTR.SHARELIB on page 49 for the authority needed to set this attribute.
- ▶ **extattr +s** marks an executable file as a program that may be loaded into a shared address space. Whether the program will actually be loaded into a shared address space depends on the setting of the environment variable `_BPX_SHAREAS`.

To specify any of the attributes, the issuer of the command needs to be the file owner or a superuser in addition to any RACF authority that may be needed. As soon as an executable file is modified, it will lose all its extended attributes.

The following example shows how the program-controlled attribute for a file can be set:

```
extattr +p /usr/sbin/inetd
```

3.3.15 z/OS UNIX kernel address space

The z/OS UNIX kernel address space is the owner of every process it forks on behalf of /etc/rc definitions. That means the user ID of the kernel address space must be defined to BPX.DAEMON with READ access.

In our sample setup, the kernel user ID is OMVSKERN. If you start InetD and syslogd via the /etc/rc shell script, they will execute under the OMVSKERN user ID.

The output of the D OMVS,A=ALL command for InetD looks like the following:

OMVSKERN INETD1	0076	687865905	1	1FI	14.48.00	1.637
LATCHWAITPID=	0	CMD=/usr/sbin/inetd /etc/inetd.conf				

3.3.16 z/OS UNIX security considerations for TCP/IP

The following topics discuss security considerations for TCP/IP address spaces, operator commands, and a number of TCP/IP applications.

TCP/IP address spaces

As mentioned before, TCP/IP is an OS/390 UNIX application itself. Consequently, the user ID a TCP/IP address space is running under needs to have a UID associated to it, and it must be UID(0). The following example shows how to assign an OMVS segment to an existing user ID for a TCP/IP address space. This user ID should also be a protected user ID (a user ID that has no password and cannot be used to log on to any interactive service):

```
ALU tcpip_user NOPASSWORD OMVS(UID(0) HOME(/) PGM(/bin/sh))
```

The TCP/IP address space is started as a started procedure, so a profile in class STARTED is required for it to pick up its user ID. The following example shows a definition:

```
RDEFINE STARTED TCPIP.** STDATA(USER(TCPIP1) GROUP(SYS1) TRUSTED(NO))  
SETROPTS RACLIST(STARTED) REFRESH
```

VARY TCP/IP command

The VARY TCPIP command is used to stop and restart TCP/IP address spaces, make changes to system operation and network configuration, start and stop devices, trace data, and control certain TCP/IP applications.

If the installation has activated class OPERCMDS in RACF, profiles should be defined to control the use of the VARY TCPIP command. The resource names shown in Table 3-4 are in use.

Table 3-4 RACF OPERCMDS resources for the VARY.TCPIP command

VARY TCPIP command options	RACF resources
VARY TCPIP,,DATTRACE	MVS.VARY.TCPIP.DATTRACE
VARY TCPIP,,DROP	MVS.VARY.TCPIP.DROP
VARY TCPIP,,OBEYFILE	MVS.VARY.TCPIP.OBEYFILE
VARY TCPIP,,PKTTRACE	MVS.VARY.TCPIP.PKTTRACE
VARY TCPIP,,START	MVS.VARY.TCPIP.START
VARY TCPIP,,STOP	MVS.VARY.TCPIP.STOP

The following example shows the definition of profiles for DROP and OBEYFILE. Users should be authorized with access level CONTROL. The class needs to be refreshed after the changes.

```
RDEFINE OPERCMDS (MVS.VARY.TCPIP.DROP) UACC(NONE)  
RDEFINE OPERCMDS (MVS.VARY.TCPIP.OBEYFILE) UACC(NONE)  
PERMIT MVS.VARY.TCPIP.DROP ACCESS(CONTROL) CLASS(OPERCMDS) ID(uid)  
PERMIT MVS.VARY.TCPIP.OBEYFILE ACCESS(CONTROL) CLASS(OPERCMDS) ID(uid)  
SETROPTS RACLIST(OPERCMDS) REFRESH
```

In many cases, it will not be necessary to define individual profiles because the different commands will be issued by the same operators. In this case it is adequate to define a single generic profile that covers all commands. The following example shows this; CONTROL access is given to group NETOPER:

```
RDEFINE OPERCMDS (MVS.VARY.TCPIP.** ) UACC(NONE)  
PERMIT MVS.VARY.TCPIP.** ACCESS(CONTROL) CLASS(OPERCMDS) ID(NETOPER)
```

3.3.17 IBM-supplied daemons

A number of daemon programs are supplied with z/OS. These daemons require some security considerations, which we discuss in the following sections. All the daemons discussed here are z/OS UNIX programs, so the user ID they are running under must have a UID assigned. Please note that some daemons also have an MVS, non-UNIX equivalent, which will not be discussed here.

z/OS UNIX supplies these daemons:

- ▶ InetD, the Internet daemon
- ▶ rlogind, the remote login daemon
- ▶ cron, the batch scheduler
- ▶ lm, the Communications Server login monitor
- ▶ uucpd, the UUCP daemon

The InetD, rlogind, cron, lm, and uucpd programs reside in the HFS directory /usr/sbin. They have the sticky bit turned on (see “The sticky bit” on page 51).

InetD

InetD is a generic listener program that is responsible for starting server programs at the client's request. The operation of InetD is controlled through its configuration file /etc/inetd.conf.

If InetD is forked by /etc/rc, it will inherit the user ID of the BPXOINIT process, which usually should be OMVSKERN. This user ID is a superuser and has access to BPX.DAEMON in class FACILITY. If InetD is started from the UNIX shell, it needs to be started by a superuser.

If InetD is started as a started procedure, a profile in class STARTED is needed to assign a user ID to InetD.

When InetD is running and receives a request for a connection, it processes that request according to the port number over which it received the request. If, for example, a user tries to log in from a remote system using rlogin, InetD receives the request at port number 513. The correlation between port numbers and service names as they are specified in /etc/inetd.conf is established through your /etc/services file, where you define which port number is associated with a service name. After having received the connection request, InetD issues a fork(), and the forked process issues an exec() to start the requested server program (in this case, the rlogin program), which then runs as the server.

The user ID under which the new process will initially start is defined in /etc/inetd.conf. The figure below shows the definitions of the services in /etc/inetd.conf. We have extracted the Telnet and rlogin definition lines.

```
=====
service | socket | protocol | wait/ | user | server | server program
name    | type   |          | nowait|     | program| arguments
=====
otelnet stream tcp nowait TCPIP3 /usr/sbin/otelnetd otelnetd -l
login   stream tcp nowait OMVSKERN /usr/sbin/rlogind rlogind -m
=====
```

In this example, we have assigned the user ID OMVSKERN to rlogind and TCPIP3 to telnetd.

telnetd, RSHD, REXECD and rlogind

The most often-used services that are managed by InetD are Telnet, RSH, REXEC, and rlogin. We briefly describe one service here, the telnetd daemon.

InetD listens for Telnet connection requests usually on port 23, although a different port can be used. Immediately after such a request arrives, InetD uses the fork() service to create a child process and starts the telnetd server program as specified in /etc/inetd.conf.

The telnetd server program enters the initial session setup window with the Telnet client and requests a user ID and password from the client end user. In this phase, the user ID and password are checked. After these are accepted by the current phase of telnetd, telnetd restarts itself via a new exec() call with a new set of parameters and in addition, a shell process is started.

Both the telnetd process and the shell process execute under the end user ID, and not the original daemon user ID. The shell process has the telnetd process as parent, the telnetd process has the InetD process as parent, and InetD in turn has the highest level process of all, the BPXOINIT process, as parent.

Processing for the other interactive services, such as RSHD, REXECD, and rlogind, is similar.

All servers need to run under the superuser authority. Furthermore, RSHD and REXECD require daemon authority to change the user identity of its child processes. If they are executed without daemon authority, an rsh/rexec client program will fail and error messages similar to the following will show up in the MVS console:

```
ICH408I USER(TCPIP3 ) GROUP(OMVSGRP ) NAME(TCPIP TASKS )
        BPX.DAEMON CL(FACILITY)
        INSUFFICIENT ACCESS AUTHORITY
        ACCESS INTENT(READ ) ACCESS ALLOWED(NONE )
```

Because all those servers are invoked by InetD, the user ID associated with InetD should have the access to BPX.DAEMON in most installations. The user ID OMVSKERN, which is the default user ID associated with z/OS UNIX kernel, would be recommended.

If you choose to have a daemon program run with a user ID other than OMVSKERN, the selected user ID needs to be defined with UID(0) and it needs READ access to profile BPX.DAEMON in class FACILITY.

Routing daemons

On z/OS IP there are two routing daemons:

- ▶ ORouted

ORouted is an IP routing daemon that implements RIP-1 and RIP-2. It creates and maintains network routing tables. ORouted provides an alternative to static routing protocols. ORouted determines if a new route has been established or if a route is temporarily unavailable.

1. OpenEdition Multiprotocol Route Application (OMPROUTE)

OMPROUTE is an IP routing daemon that supports RIP-1, RIP-2 and OSPF protocols. OMPROUTE was introduced in OS/390 eNetwork Communications Server V2R6 IP and is the recommended routing daemon application for z/OS IP. It provides the technology to determine the most efficient route for use in the autonomous system. OSPF is the preferred protocol in OMPROUTE.

Note: ORouted and OMPROUTE will not run on the same TCP/IP stack.

RACF control of who can start OMPROUTE or ORouted was integrated into CS for OS/390 V2R8 IP. If the RACF profile is not defined, no additional security checks are performed.

To enable ORouteD and OMPROUTE RACF authorization for a particular user ID, enter the following commands with the ID of the user or with the ID of the started task to be authorized:

```
RDEFINE OPERCMDS (MVS.ROUTEMGR.OMPROUTE) UACC(NONE)
PERMIT MVS.ROUTEMGR.OMPROUTE ACCESS(CONTROL) CLASS(OPERCMS) ID(userid)
RDEFINE OPERCMDS (MVS.ROUTEMGR.OROUTED) UACC(NONE)
PERMIT MVS.ROUTEMGR.OROUTED ACCESS(CONTROL) CLASS(OPERCMS) ID(userid)
SETROPTS RACLIST(OPERCMS) REFRESH
```

If neither the user nor the started task is RACF authorized for ORouteD, the following message is displayed:

```
EZZ5020E "User not RACF authorized to start OE Routed"
```

If neither the user nor the started task is RACF authorized for OMPROUTE, the following message is displayed:

```
EZZ5020E "User not RACF authorized to start OMPROUTE"
```

3.3.18 MVS sockets server applications

In CS for OS/390 Release 8, the only server that ran in a traditional MVS environment was the TN3270 server, which ran within the TCP/IP address space. All others needed an OMVS segment defined in the RACF database, except those that use the PASCAL API. The PASCAL servers are LPD, SMTP, and DNS.

The following application servers are covered in Part 4, "Application security" on page 183 of this book:

- ▶ IBM HTTP Server
- ▶ TN3270 server
- ▶ FTP server
- ▶ TFTP server
- ▶ NFS server
- ▶ UNIX rlogin/rsh/rexecd
- ▶ SNMP

3.3.19 Summary

Table 3-5 gives a summary of all the definitions you have to create to run UNIX System Services servers with MVS-like security.

Table 3-5 Overview of processes, UIDs and RACF FACILITY class profiles

Process/Job	UID	BPX.SERVER access	BPX.DAEMON access
OMVS	0	NONE	READ
TCP/IP system address space	0	NONE	NONE
InetD	0	NONE	READ
syslogd	0	NONE	NONE
FTPD	0	NONE	NONE
Client User	Not 0	NONE	NONE
Web Server	0	READ	NONE

Process/Job	UID	BPX.SERVER access	BPX.DAEMON access
Web User	Web User not 0, not UID of administrator, not in same group	NONE	NONE

Note: Remember that as soon as you have defined BPX.DAEMON, program control must have been set up correctly and must be active. If not, all server requests are going to fail with various error messages. Whenever something does not work as expected, make it a rule to look at your z/OS system log to see if there are any security violation messages that might indicate a security definition problem.

You may also wish to investigate the sample job that will assist you in performing some of these tasks. The job can be found in member EZARACF, shipped in hlq.SEZAINST, which contains sample RACF commands for most TCP/IP-related security functions.



TCP/IP stack resource access

This chapter discusses the SAF security profiles that can be used to protect access to a TCP/IP stack's resources.

The stack resources being secured can be categorized into the following:

- ▶ Access to use a specific TCP/IP stack on a z/OS image is discussed in 4.1, "TCP/IP stack access control" on page 60
- ▶ Access to send packets to IP hosts from a specific TCP/IP stack is discussed in 4.2, "Network access control" on page 67
- ▶ Access for a server to bind() to a port on a specific TCP/IP stack is discussed in 4.3, "Port Access control" on page 70

SAF profiles using the SERVAUTH class are used to control all of these access types.

4.1 TCP/IP stack access control

This section discusses the SAF security profiles that can be used to protect access to a TCP/IP stack's resources.

The z/OS images are capable of being configured to support more than one concurrent instance of TCP/IP. In order to support more than one TCP/IP stack, the Physical File System must be configured to support Common INET (C-INET).

Each stack, in effect, acts like a different host on an IP network, with IP routes possible between the individual stacks. Before V2R10, multiple stacks were sometimes used so that two or more servers that bound to the same port number and protocol (with connection requests from any interface) could run simultaneously. One example is the OE Telnet server and the TN3270 server. This is no longer necessary as the BIND keyword on the PORT statement can convert the server's bind() to use a specific IP address rather than the "don't-care" (INADDR_ANY) address on a per-server basis. This just means that each server is differentiated from the others that use the same port by the client targeting a different IP address. However, if you must implement multiple stacks for a different reason, such as having a production stack and a test stack, Stack Access security may be beneficial for you.

Whether your system has a single stack or multiple stacks, you may want to control which users/servers have access to which (or the only) stack.

4.1.1 Stack Access overview

Stack Access control provides a way to permit or deny users or groups of users access to a TCP/IP stack. The function controls the ability of a user to open an IP socket, with the socket() API function.

Stack access control is implemented by defining a SERVAUTH class profile to the SAF product (RACF or equivalent) on your system. To allow for SAF databases to be shared across multiple z/OS images and for multiple stacks within each of those images, the profile name is in the format:

```
EZB.STACKACCESS.sysname.tcpipname
```

Where:

- ▶ EZB.STACKACCESS is constant
- ▶ sysname is the name of the z/OS image in the sysplex (&SYSNAME symbol)
- ▶ tcpipname is the TCP/IP job name to be controlled

There are no TCP/IP configuration statements required.

If the system administrator does not define the SAF profile to protect the stack, then stack access control is not performed and all users have access to the stack. If there are multiple stacks in the z/OS system, each stack can be enabled for stack access control independently.

The security checking will be bypassed for socket() calls originating in the TCP/IP, UNIX System Services or VTAM address spaces. If the EZB.STACKACCESS.sysname.tcpipname profile is enabled for the stack, all non-system IP code must be running under a user ID that is permitted to that SAF profile.

4.1.2 Example setup

This example shows how to control access to the TCP/IP stack running with a job name of TCPIPC on the z/OS system SC63. The first thing to do is to add the SERVAUTH STACKACCESS profile to RACF or your equivalent SAF product. As mentioned, the format of the profile name is EZB.STACKACCESS.sysname.tcpname, so in our example the profile name will be EZB.STACKACCESS.SC63.TCPIPC as shown in Figure 4-1.

```
RACF - GENERAL RESOURCE SERVICES - ADD
OPTION ==>

ENTER THE FOLLOWING PROFILE INFORMATION:

CLASS      ==> SERVAUTH

PROFILE    ==> EZB.STACKACCESS.SC63.TCPIPC

                                <==end of data

USE A MODEL      ==>          YES or NO
```

Figure 4-1 RACF SERVAUTH profile for TCP/IP stack access

If you are using the RACF ISPF panels to set up the profile, then the next panel, shown in Figure 4-2, is displayed. This is where you specify that Universal Access (UACC) is none, meaning the default for any user is to be denied access.

```
RACF - ADD GENERAL RESOURCE PROFILE
COMMAND ==>

CLASS:      SERVAUTH
PROFILE    _ EZB.STACKACCESS.SC63.TCPIPC

ENTER OR CHANGE THE FOLLOWING INFORMATION:

OWNER              ==> FOCAS      Userid or group name
LEVEL              ==> 0          0-99
FAILED ACCESSES   ==> FAIL       FAIL or WARN
UACC               ==> NONE       NONE, READ, UPDATE,
                                CONTROL, ALTER or EXECUTE
AUDIT SUCCESSES   ==> NOAUDIT    READ, UPDATE, CONTROL,
                                ALTER, or NOAUDIT
AUDIT FAILURES    ==> READ       READ, UPDATE, CONTROL,
                                ALTER, or NOAUDIT
NOTIFY            ==>           Userid

TO ADD OPTIONAL INFORMATION, ENTER YES      ==>
```

Figure 4-2 RACF SERVAUTH profile for TCP/IP stack access - second panel

Once the profile has been added (and if using RACF, a **setropts raclist(servauth) refresh** is performed), the stack is now protected against unauthorized use.

Figure 4-3 shows the RACF error message that is returned when an unauthorized user attempts to access a TCP/IP stack without having the appropriate access to the STACKACCESS profile.

```

ICH408I USER(FOCAS  ) GROUP(SYS1  ) NAME(PETER FOCAS IBM NZ  ) 662
EZB.STACKACCESS.SC63.TCPIPC CL(SERVAUTH)
INSUFFICIENT ACCESS AUTHORITY
ACCESS INTENT(READ  ) ACCESS ALLOWED(NONE  )

```

Figure 4-3 RACF error returned from unauthorized TCP/IP stack access

In this case the TSO NETSTAT DEV command was issued, but it could have been any TCP/IP command (**ping**, **tracerte**, etc.)

Now we authorize the user to access the TCP/IP stack and reissue the command. In RACF, you add a user, or group to an “access-list” for the profile. Figure 4-4 shows the RACF ISPF panel to add the previous user to the access list for the EZB.STACKACCESS.SC63.TCPIPC profile. Once again, if RACF is being used, a **setropts raclist(servauth) refresh** command will be required.

```

RACF - MAINTAIN GENERAL RESOURCE ACCESS LIST - ADD
COMMAND ==>

CLASS:          SERVAUTH
PROFILE         _ EZB.STACKACCESS.SC63.TCPIPC

Enter the access authority to be granted:

AUTHORITY      ==> read      NONE, READ, UPDATE,
CONTROL, ALTER or EXECUTE

Enter the users or groups for which entries are to be added:

==> focas      ==>          ==>          ==>          ==>
==>          ==>          ==>          ==>          ==>

```

Figure 4-4 RACF ISPF panel to add user to access list

Once a user has been added to the access list for the STACKACCESS profile, he or she is authorized to access the stack and enter TCP/IP commands, as shown in Figure 4-5 on page 63.

```

Enter TSO or Workstation commands below:

===> netstat dev

=>
MVS TCP/IP NETSTAT CS V1R2          TCPIP NAME: TCPIPC          17:19:29
DevName: LOOPBACK                  DevType: LOOPBACK  DevNum: 0000
  DevStatus: Ready
  LnkName: LOOPBACK                 LnkType: LOOPBACK  LnkStatus: Ready
    NetNum: 0  QueSize: 0
    BytesIn: 41961                   BytesOut: 41961
  BSD Routing Parameters:
    MTU Size: 00000                   Metric: 00
    DestAddr: 0.0.0.0                 SubnetMask: 0.0.0.0
  Multicast Specific:
    Multicast Capability: No

***

```

Figure 4-5 The successful TCP/IP command once authorized for stack access

We have just covered enabling a user to access the TCP/IP stack. This concept of a user applies equally to the owner of any server running on the stack. For example, the FTP started task SAF owner would have to be given access to the stack's RACF profile.

If you have generic servers (see "Generic servers" on page 63) such as the BIND 9.0-based DNS server running in your system, be aware that they will try to bind() to all home IP addresses defined on any TCP/IP stack on the z/OS image. If at least one stack is protected by a STACKACCESS profile, the generic servers RACF user ID will have to have READ access to the profile(s), or they will have to be made non-generic servers.

```

ICH408I USER(STC      ) GROUP(SYS1      ) NAME(STARTED TASK      )
      EZB.STACKACCESS.SC63.TCPIPC CL(SERVAUTH)
      INSUFFICIENT ACCESS AUTHORITY
      ACCESS INTENT(READ  ) ACCESS ALLOWED(NONE  )

```

Figure 4-6 RACF error when generic server does not have READ access to the stack

Figure 4-6 shows an example of a generic server attempting to bind to an IP address in a stack protected with a STACKACCESS profile. The assumption here is that the DNS server runs under RACF user "STC", and that the stack that is protected is TCPIPC running on z/OS system SC63. In this case, the error message was repeated at one-minute intervals until the "STC" user was given READ access to the EZB.STACKACCESS.SC63.TCPIPC RACF profile. Note that the DNS server in this example was able to bind successfully to other stacks on the system even though RACF did not allow access to the TCPIPC stack. To make a server non-generic, see the discussion in 4.1.3, "Transport/stack affinity" on page 63.

4.1.3 Transport/stack affinity

Generic servers

The C-INET environment (multiple TCP/IP stacks per z/OS image) has some implications for certain servers. Servers accept inbound connections by listening for connection requests on a particular protocol, port and IP address. Some servers, termed "generic" servers, do not listen for connection requests sent to a particular IP address, but accept connection requests received on any active interface (in the API this is termed binding to INADDR_ANY). This

means a generic server accepts connection requests from *any home IP address defined within any active TCP/IP stack on a z/OS image*. If a server issues a bind to a specific IP address or uses `setibmopt()` to choose a transport provider, then that server is no longer considered generic.

The following daemons supplied with CS for z/OS are generic applications:

- ▶ FTPD
- ▶ z/OS UNIX RSHD
- ▶ z/OS UNIX REXECD
- ▶ z/OS UNIX telnetd
- ▶ z/OS UNIX SENDMAIL
- ▶ z/OS UNIX POPPER
- ▶ TFTPd
- ▶ TIMED
- ▶ z/OS UNIX Portmap

z/OS UNIX RSHD, REXECD and telnetd are usually started by the INETD daemon, which is shipped as part of the z/OS UNIX. Because INETD is also a generic daemon, any server processes started by INETD inherently become generic servers as well.

Taking the FTP server as an example, if we start the FTP daemon and there are three active TCP/IP stacks, all three stacks will be able to handle incoming FTP connection requests. See Figure 4-7 for a diagrammatic representation.

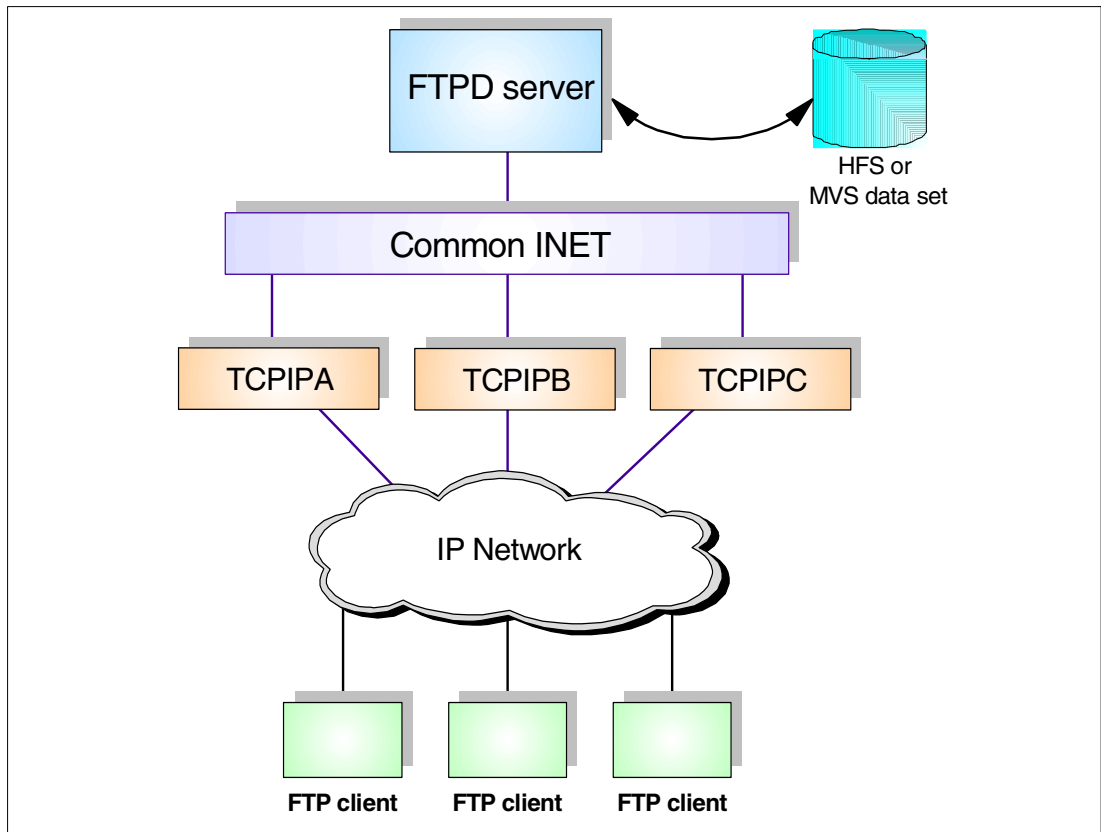


Figure 4-7 Diagram showing how an FTP server accepts connections from all TCP/IP stacks

Note that pointing a generic server to a particular TCPIP.DATA file (which includes the TCPIPJOBNAME statement) does not (see note below) alter the fact that the server has bound a socket for any active home IP address. Any active stack can accept an incoming request for a generic server and pass it via C-INET to the waiting server. This could be a security exposure for customers who do not want an application's services available on all TCP/IP stacks. Or, customers may want FTPA connected to, say, TCPIPA and FTPB connected to TCPIPB.

Note: Indirectly, an application can use the `__iptcpn()` C language function call to determine the TCPIPJOBNAME and then use `setibmopt()` to bind to the transport.

If you are implementing stack access control (see 4.1, "TCP/IP stack access control" on page 60) to enforce which stack users can access, you should bear in mind that a generic server essentially attempts to access all active stacks. If the server's RACF ID is not permitted access to the STACKACCESS profiles of every stack (if defined), a RACF error will occur. If that is not acceptable to you, some method is required to ensure that a server binds to a specific IP address (as opposed to INADDR_ANY) and therefore to a specific instance of TCP/IP. The mechanism to enable this is referred to as *transport affinity* (sometimes, the term stack affinity is also used).

The `_BPXK_SETIBMOPT_TRANSPORT` environment variable, when set, has an effect similar to the `setibmopt()` function call provided by the C/C++ compiler and described in the *z/OS C/C++ Run-Time Library Reference*. This variable can be set in the JCL for a started procedure or batch job that executes a z/OS UNIX C/C++ program to indicate which TCP/IP stack instance the application should bind to. TCP/IP applications that require affinity to a specific TCP/IP stack, such as OSNMPD and OROUTED, use the `setibmopt()` function call directly. For other programs, the BPXTCAFF program (as a job step) can be executed to set transport affinity for an entire address space. For more details on BPXTCAFF (and other transport options), see *z/OS V1R2.0 UNIX System Services Planning*, GA22-7800.

Transport affinity using `_BPXK_SETIBMOPT_TRANSPORT`

In Example 4-1, the ENVAR parameter would instruct the FTP daemon to bind to the TCPIPC stack only. If we ran this server with the `_BPXK_SETIBMOPT_TRANSPORT` variable set as shown, only FTP requests arriving on an IP address controlled by stack TCPIPC would be serviced by this instance of the server.

Example 4-1 `_BPXK_SETIBMOPT_TRANSPORT` to set stack affinity in an FTP started task

```
//FTPDC  PROC MODULE='FTPD',PARMS='',
//          P1='ENVAR("_BPXK_SETIBMOPT_TRANSPORT=TCPIPC")'
//FTPDC  EXEC PGM=&MODULE,REGION=0K,TIME=NOLIMIT,
//          PARM='POSIX(ON) ALL31(ON) &P1/&PARMS'
//CEEDUMP DD SYSOUT=*
//SYSTCPD DD DISP=SHR,DSN=TCPIPC.&SYSNAME..TCPPARMS(TCPDATC)
//SYSFTSX DD DISP=SHR,DSN=TCPIPMVS.STANDARD.TCPXLBIN
```

Essentially, using `_BPXK_SETIBMOPT_TRANSPORT` causes the LE runtime environment to issue a `setibmopt()` call on behalf of the executing program.

Once stack affinity has been established for a server, it will only be presented with connection requests sent to an IP HOME address in that stack, as depicted in Figure 4-8 on page 66.

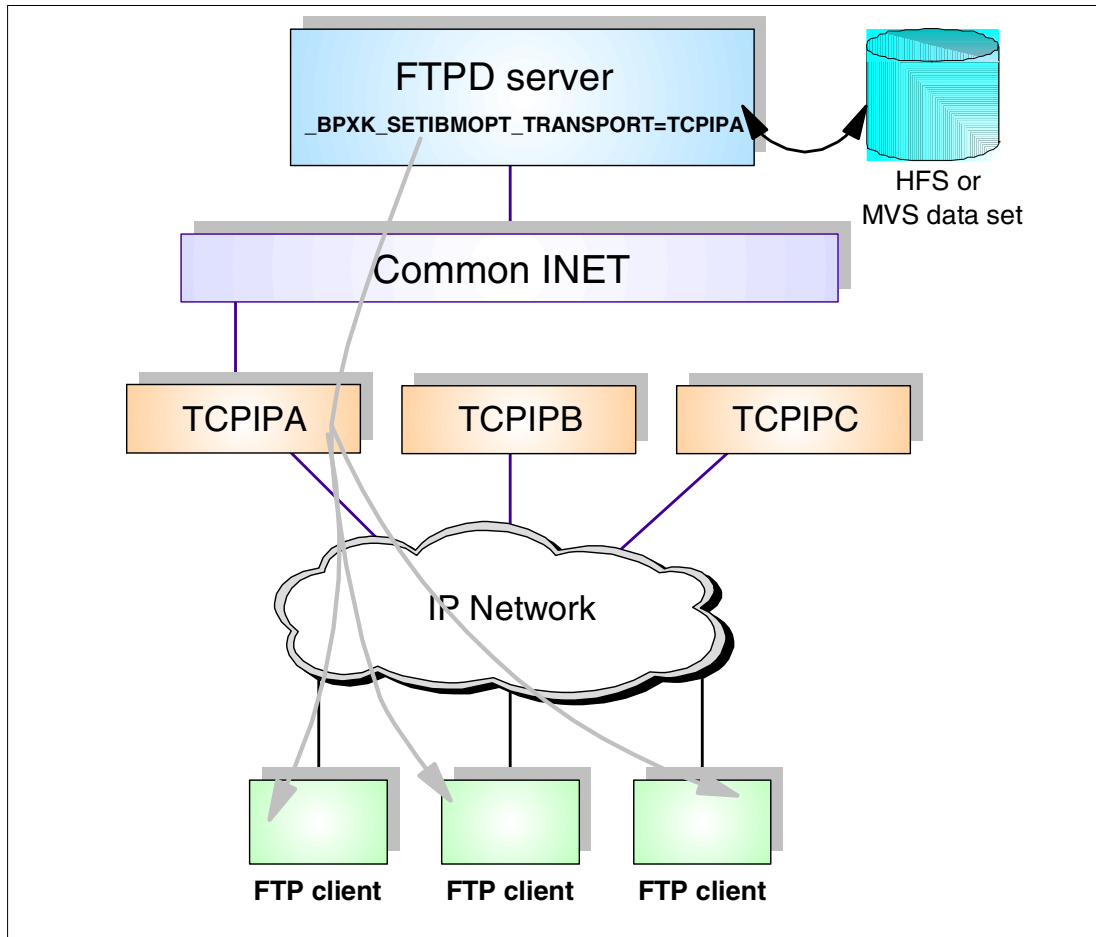


Figure 4-8 FTP Server with transport affinity set to TCPIPA

Transport affinity using BIND keyword on PORT statement

A second and more granular option for indirectly controlling the transport affinity is the BIND keyword on a PORT statement in the TCP/IP profile data set, as shown in Example 4-2.

Example 4-2 PORT statement BIND keyword to set IP address for server

```
PORT
23 TCP INETD1 BIND 9.12.6.62
```

This statement forces otelnetd (started by INETD1), a generic application, to bind to IP address 9.12.6.62, effectively establishing transport affinity. Note that using the BIND keyword is a much more granular control than using the environment variable `_BPXK_SETIBMOPT_TRANSPORT` or the `setibmopt()` call. When binding to a specific IP address, only that IP address can be used as an endpoint of communication. When using `setibmopt()`, any available IP address in the TCP/IP stack chosen can be a communication endpoint.

4.2 Network access control

Network access control enables system administrators to represent access to an IP network, subnetwork or host as a SAF resource (RACF or equivalent). The ability to send IP packets to those networks, subnetworks, or hosts can then be permitted or denied at a SAF user or group level. This feature provides an additional layer of security to any authentication or authorization that is used at the target system. It might be used, for example, to prevent access to the Internet by anyone except the SMTP server, or it could be used to stop general users attempting to Telnet to a server that contained payroll information.

Important: Network access control cannot prevent a user from receiving packets from any particular network address. It controls only the *transmission* of packets to particular networks, subnetworks, or hosts. Inbound protection, if needed, would normally be provided by some sort of firewall.

4.2.1 Network access control overview

In the TCP/IP profile, there is a parameter block, NETACCESS/ENDNETACCESS. This is where you specify the mapping of an IP network, subnetwork, or host to a SAF profile. Example 4-3 shows a sample NETACCESS block.

Example 4-3 NETACCESS statements

```
NETACCESS
  9.24.104.0/24      MYSUBNET      ;my workstation subnet
  9.24.104.119/32   MYPC          ;my workstation
  DEFAULT 0         WORLD          ;everything else
ENDNETACCESS
```

In this example, access to hosts on subnet 9.24.104 is mapped to SAF profile MYSUBNET, access to host 9.24.104.119 is mapped to SAF profile MYPC, and access to any other host is mapped to SAF profile WORLD. These SAF profiles are defined to RACF in the SERVAUTH class. The complete profile name to be defined is in the following format:

EZB.NETACCESS.systemname.stackname.resourcename

Tip: On the NETACCESS statement, there are two ways to specify the subnet mask. One way is the traditional decimal notation, such as 255.255.255.0. The second way is to use a number, up to 32, that specifies the number of bits, left to right, that should be used as a subnet mask if the mask is expressed in binary. For example, the subnet mask 255.255.255.0 expressed in binary is 11111111.11111111.11111111.00000000. Note that there are 24 bits set on. To specify this particular mask on a NETACCESS statement, you could use either of the following two ways, using the IP address 192.168.100.0 as an example:

```
NETACCESS 192.168.100.0 255.255.255.0
```

or

```
NETACCESS 192.168.100.0/24
```

The system that we used in Example 4-3 was on a z/OS image named SC63 with a TCP/IP stackname of TCPIPC. Therefore, the three profiles that need to be defined are:

1. EZB.NETACCESS.SC63.TCPIPC.MYSUBNET

2. EZB.NETACCESS.SC63.TCPIPC.MYPC
3. EZB.NETACCESS.SC63.TCPIPC.WORLD

If you define these profiles to RACF with UACC(NONE), then users must be specifically permitted access to these profiles in order to send IP packets to the address(es) represented by the profiles.

If a user is attempting to send an IP packet to a host that is not covered by any network/mask entry in the NETACCESS block, access is automatically allowed. However, if a DEFAULT statement is present, then access is granted or denied based on the user's access to the SAF profile mapped by the DEFAULT statement.

4.2.2 Server considerations

End users are not the only users of TCP/IP to be affected by NETACCESS control. Any IP applications (servers) that run under their own user IDs would need access to the SAF profiles of the desired networks, if these networks are protected by a NETACCESS statement. For example, a server such as the FTP daemon would need to be permitted access to any hosts or subnets that an FTP transfer will be performed with.

There is another subtlety to be considered. If you have a user out on the network that wants to FTP to/from your FTP server, then the user ID that this user logs on with must have been permitted to NETACCESS if their own IP address or network is protected.

Remember, if you protect the DNS server(s) with NETACCESS, just about all users and servers will need access to this. It might be easiest to define the NETACCESS profiles for the DNS servers with a UACC(READ) so that everyone is permitted access by default.

4.2.3 Using NETSTAT for Network Access control

The console command **D TCPIP,stackname,Netstat,ACcess,NETWork** shows how IP addresses/masks are mapped to SAF profiles. See Figure 4-9 for the NETSTAT console command to display the network access control for the test TCPIPC system.

```
D TCPIP,TCPIPC,NETSTAT,ACCESS,NETWORK
EZZ2500I NETSTAT CS V1R2 TCPIPC 451
NETWORK ACCESS INFORMATION
NETWORK PREFIX ADDRESS MASK SAF NAME
DEFAULT 0.0.0.0 WORLD
9.24.104.0 255.255.255.0 MYSUBNET
9.24.104.119 255.255.255.255 MYPC
3 OF 3 RECORDS DISPLAYED
```

Figure 4-9 NETSTAT command to display network access control in our test system

The TSO NETSTAT command does not display this information.

4.2.4 Working example of Network Access control

We implement network access control on the TCP/IP stack discussed in 4.2.1, “Network access control overview” on page 67. In that section we show the three SAF profile names that need to be defined for the given NETACCESS block. Example 4-4 on page 69 shows the configuration again.

Example 4-4 NETACCESS block used in examples

```
NETACCESS
 9.24.104.0/24      MYSUBNET      ;my workstation subnet
 9.24.104.119/32   MYPC           ;my workstation
 DEFAULT 0         WORLD           ;everything else
ENDNETACCESS
```

Once the SAF profiles names shown in 4.2.1, “Network access control overview” on page 67 have been defined to RACF, we attempt to use the TSO PING command to send a packet to the workstation 9.24.104.119. Since we have not been permitted access to the 9.24.104.119 host (profile MYPC), you would expect an error. Figure 4-10 shows the error message from RACF indicating that access to the MYPC profile was not permitted.

```
ICH408I USER(FOCAS ) GROUP(SYS1 ) NAME(PETER FOCAS IBM NZ ) 331
EZB.NETACCESS.SC63.TCPIP.MYPC CL(SERVAUTH)
INSUFFICIENT ACCESS AUTHORITY
ACCESS INTENT(READ ) ACCESS ALLOWED(NONE )
```

Figure 4-10 RACF error while attempting PING to host covered by host profile

Note: Even though both the host 9.24.104.119, and its subnet 9.24.104.0 are protected by SAF profiles, the SAF check is only performed on the *most specific* network/host entry. An easier way to say this is that the entries in the NETACCESS block are checked starting with those with the most bits specified in the subnet mask first, until a match is found.

If we now attempt to ping a new host 9.24.104.20, we would expect a RACF error when the TCP/IP address space did a SAF check for access to the MYSUBNET profile, as this is the most specific entry for that host address. We attempt a ping to 9.24.104.120 and get the error as expected. Figure 4-11 shows this.

```
ICH408I USER(FOCAS ) GROUP(SYS1 ) NAME(PETER FOCAS IBM NZ ) 442
EZB.NETACCESS.SC63.TCPIP.MYSUBNET CL(SERVAUTH)
INSUFFICIENT ACCESS AUTHORITY
ACCESS INTENT(READ ) ACCESS ALLOWED(NONE )
```

Figure 4-11 RACF error while attempting to ping a host covered by subnet profile

Lastly, we show an example of the error you would receive when attempting network access to a host not specified on any NETACCESS statements. This assumes you have coded a DEFAULT statement in the NETACCESS block. If you do not, access permission is not checked for any host not covered by any other NETACCESS statement. We tried to ping 192.168.10.10, an IP address that is not specifically stated in a host or subnet entry. The DEFAULT NETACCESS statement is therefore used for the SAF check that is mapped to profile WORLD. As shown in Figure 4-12, we got a RACF error message indicating we did not have access to EZB.NETACCESS.SC63.TCPIP.WORLD.

```
ICH408I USER(FOCAS ) GROUP(SYS1 ) NAME(PETER FOCAS IBM NZ ) 449
EZB.NETACCESS.SC63.TCPIP.WORLD CL(SERVAUTH)
INSUFFICIENT ACCESS AUTHORITY
ACCESS INTENT(READ ) ACCESS ALLOWED(NONE )
```

Figure 4-12 RACF error while attempting to ping a host covered by the DEFAULT statement

4.3 Port Access control

The ability of a server to bind to a specific port can be controlled in a number of ways using the UDPCONFIG, TCPCONFIG and PORT (or PORTRANGE) TCP/IP profile statements.

The use of TCPCONFIG RESTRICTLOWPORTS and UDPCONFIG RESTRICTLOWPORTS is encouraged to enhance security. If these statements are present, low ports (<1024) can only be bound when at least one of the following is true:

- ▶ The bind is issued from a process with a UNIX superuser (UID 0)
- ▶ The bind is issued from an APF-authorized application
- ▶ The port is reserved for the application by job name, which may include *, OMVS, or TSO user ID
- ▶ If a SAF resource name is used, the binding process's user ID must be permitted to the resource by the security product (described later)

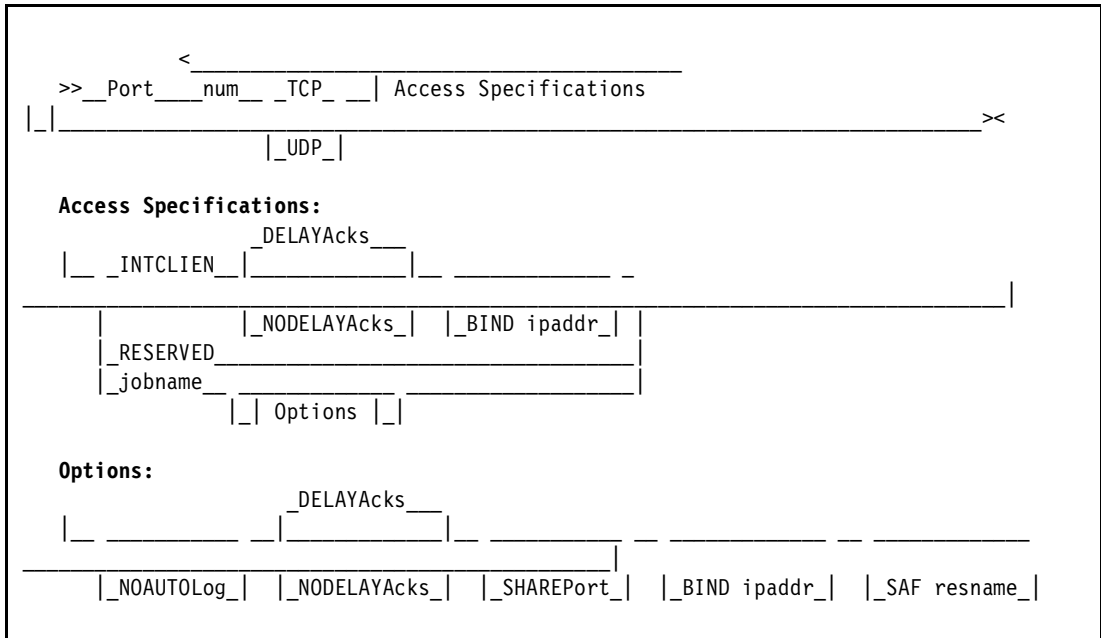


Figure 4-13 The format of the PORT configuration statement

Figure 4-13 shows the PORT/PORTRANGE statement and associated parameters.

- ▶ The RESERVED keyword will shut down a port from being used by any job name at all. The keyword can also be specified on the PORTRANGE statement. This readily allows an installation to clamp down very tightly on usage of ports if such control is desired.
- ▶ Specifying a job name on a PORT or PORTRANGE statement restricts the use of that port (and protocol) to the specified job name. Multiple PORT statements can be specified for a TCP port but not for UDP. Note that a job name of '*' (the wildcard character) is normally used with the SAF keyword, described next.
- ▶ Specifying the SAF keyword and profile name provides a mapping from a port and protocol to a SAF SERVAUTH class profile. A server attempting to bind to this port and protocol is checked for SAF access to the profile named. The use of the SAF keyword is covered in this section.

4.3.1 The PORT/PORTRANGE SAF keyword

The SAF keyword can be specified along with all other valid options on the PORT and PORTRANGE statements. The special job name wildcard '*' is normally used with the SAF keyword so that access to the port is completely handled by the server's SAF profile rather than job name. Of course, a specific job name and the keyword SAF can still be coded together if you would like to secure not only the job name that can bind a socket, but also the SAF user associated with the job.

To get a better idea of how SAF-based port access is implemented, look at the example in Figure 4-14.

```
PORT
20  TCP * SAF FTP20      ; FTP Control socket
21  TCP * SAF FTP21      ; FTP Data socket
23  TCP INTCLIEN        ; Internal TN3270 Server
512 UDP RESERVED        ; Shut down port 512
611 TCP * SAF USER611   ; User process for socket 611
```

Figure 4-14 PORT statements for stack TCPIPC

Given the PORT statements in Figure 4-14, UDP port 512 is reserved, and therefore completely unavailable for use. Any process attempting to use TCP ports 20, 21 or 611 would have to be SAF authorized. The following SERVAUTH profiles would have to be defined (assuming the system name is SC63 and the TCP/IP stack name is TCPIPC):

- ▶ EZB.PORTACCESS.SC63.TCPIPC.FTP20
- ▶ EZB.PORTACCESS.SC63.TCPIPC.FTP21
- ▶ EZB.PORTACCESS.SC63.TCPIPC.USER611

This form of port reservation might be used when a reserved low port needs to be accessed by many potential users via a client program that is not APF-authorized. All users needing the ability to run this program would have to be permitted to this RACF resource.

4.3.2 SAF keyword on FTP or any other well-known PORTs

With FTP ports 20/21 (or any well-known port usually used for a system-type server), there is a possible security exposure when permitting port use by the SAF keyword. Once a user is permitted to bind to, say, port 20, they can bind to that port using any program, not just through the FTP server. To prevent this, we recommend that you do not code a SAF keyword for port 20, but instead use the RESTRICTLOWPORTS parameter of the TCPCONFIG statement in conjunction with specifying "OMVS" as the job name for port 20. This restricts the use of port 20 to APF-authorized programs, UNIX superusers, or the FTP server.

The FTP daemon is the only user that needs to access the FTP control port 21, and hence have access to the RACF SERVAUTH profile EZB.PORTACCESS.SC63.TCPIPC.FTP21. If the FTP server does not have access to the profile mapped to port 21, you will get a RACF error similar to that shown in Figure 4-15 on page 72.

```

ICH408I USER(TCPIPMVS) GROUP(OMVSGRP ) NAME(TCPIP ) 864
EZB.PORTACCESS.SC63.TCPIPC.FTP21 CL(SERVAUTH)
INSUFFICIENT ACCESS AUTHORITY
ACCESS INTENT(READ ) ACCESS ALLOWED(NONE )
BPXF024I (TCPIPMVS) May 3 13:39:08 ftpd 84017231 : EZYFT13E bind 865
error : EDC5111I Permission denied.

```

Figure 4-15 FTP server unauthorized to use port 21

The FTP data connection port (20) is bound under the identity of the end user, not the FTP daemon. Therefore, if the PORT statement for port 20 is configured as shown in Figure 4-14 on page 71, then all end users (including the default user, if defined) who could potentially perform FTP need to be permitted to the port 20 RACF SERVAUTH profile EZB.PORTACCESS.SC63.TCPIPC.FTP20

4.3.3 Using NETSTAT to display Port Access control

The TCPIPC stack contains the PORT statement shown in in Figure 4-14 on page 71 .The console command **D TCPIP,stackname,Netstat,PORTlist** in Figure 4-16 shows the configuration of the ports, and whether a SAF profile is associated with a port (the F in the FLAGS column).

```

D TCPIP,TCPIPC,NETSTAT,PORTLIST
EZZ2500I NETSTAT CS V1R2 TCPIPC 150
PORT# PROT USER   FLAGS   RANGE      IP ADDRESS      SAF NAME
00020 TCP  *       DAF                    IP ADDRESS      FTP20
00021 TCP  *       DAF                    IP ADDRESS      FTP21
00023 TCP  TCPIPC  DA
00611 TCP  *       DAF                    IP ADDRESS      USER611
00512 UDP  RESERVED DA
5 OF 5 RECORDS DISPLAYED

```

Figure 4-16 NETSTAT PORTLIST console command display for TCPIPC

Please note that neither the TSO NETSTAT PORTLIST or the OMVS equivalent **onetstat -o** show the SAF information. This is only displayed on the z/OS console command.

4.3.4 Scenarios using port access control

To test that port access control is working as planned, the REXX sample server RSSERVER was used from *z/OS V1R2.0 CS: IP Application Programming Interface Guide*, SC31-8788. This server runs as a TSO batch job and binds to a TCP socket and waits for a client to request some lines to be returned. We are not interested in the client, since we are just testing whether the server is able to bind() to the port that is being controlled with the SAF keyword. The port number used by the server was set to 611, and in the first test, no authorization was granted to the server. The expected error of not authorized was returned as indicated in the REXX EXEC output shown in Figure 4-17 on page 73. The return code of 13 can be seen in the *z/OS V1R2.0 CS: IP Application Programming Interface Guide*, SC31-8788 as Permission denied, caller not authorized.


```

READY
%EZARXRSS
RSSERVER: Initializing
RSSERVER: Initialized: ipaddress=9.12.6.61 port=611
===> Error: RXSSRV032E SOCKET(BIND) rc=13
READY
END

```

Figure 4-17 Output from REXX server attempting to bind to port 611

The system log from the batch server job can be seen in Figure 4-18. The RACF error message indicates the batch job user does not have access to SERVAUTH profile EZB.PORTACCESS.SC63.TCPIPC.USER611. The server was attempting to bind to port 611 and port 611 maps to profile USER611.

```

IRRO10I  USERID FOCAS   IS ASSIGNED TO THIS JOB.
ICH70001I FOCAS   LAST ACCESS AT 08:25:24 ON FRIDAY, MAY 3, 2002
$HASP373 FOCAS1  STARTED - INIT 3   - CLASS A - SYS SC63
IEF403I  FOCAS1  - STARTED - TIME=08.52.16 - ASID=001B - SC63
ICH408I  USER(FOCAS   ) GROUP(SYS1   ) NAME(PETER FOCAS IBM NZ ) 902
  EZB.PORTACCESS.SC63.TCPIPC.USER611 CL(SERVAUTH)
  INSUFFICIENT ACCESS AUTHORITY
  ACCESS INTENT(READ  ) ACCESS ALLOWED(NONE  )
-
- --TIMINGS (MINS.)--
-JOBNAME  STEPNAME  PROCSTEP  RC  EXCP  CPU  SRB  CLOCK  SERV  PG
-FOCAS1   SERVER    32      82   .00   .00   .00  8833   0
IEF404I  FOCAS1  - ENDED - TIME=08.52.16 - ASID=001B - SC63

```

Figure 4-18 System log display from failing server batch job

The next step was to grant SAF access to the user associated with the batch server job, to profile EZB.PORTACCESS.SC63.TCPIPC.USER611. This is shown in Figure 4-19.

```

RACF - MAINTAIN GENERAL RESOURCE ACCESS LIST - ADD
COMMAND ===>

CLASS:          SERVAUTH
PROFILE         _ EZB.PORTACCESS.sc63.TCPIPC.user611

Enter the access authority to be granted:

AUTHORITY      ===> read      NONE, READ, UPDATE,
                CONTROL, ALTER or EXECUTE

Enter the users or groups for which entries are to be added:

===> focas     ===>          ===>          ===>          ===>
===>          ===>          ===>          ===>          ===>

```

Figure 4-19 RACF ISPF panel to grant access to PORTACCESS profile

After the RACLIST was refreshed, and the TCP/IP region recycled, the server was able to bind to socket 611 with no errors.



Operations and administration security

This chapter discusses the ways to control the use of the TCP/IP system administration commands. These commands are categorized by where they originate.

From the z/OS console, you can use the DISPLAY and VARY command for the TCP/IP address space(s). The VARY TCPIP command can be used to stop and start TCP/IP interfaces, reload configuration parameters, start and stop traces, drop TCP connections and quiesce the TN3270 server. This command is very powerful and should only be authorized to operators and/or system administrators. Protection for the VARY TCPIP command is covered in 5.1, “z/OS VARY TCPIP command security” on page 76.

From TSO, there is the NETSTAT command, and from the UNIX shell there is the **onetstat** command. Both of these commands are used primarily to display information about the local TCP/IP environment. You may want to restrict these commands so that people cannot obtain information about your TCP/IP configuration, perhaps in preparation for an attack of some kind. Protection of the TSO NETSTAT and OMVS **onetstat** commands are covered in 5.2, “TSO NETSTAT and UNIX onetstat command security” on page 79.

5.1 z/OS VARY TCPIP command security

This section describes the mechanisms by which you can limit users to the VARY TCPIP commands.

5.1.1 RACF profile details

The z/OS console VARY TCPIP commands are protected with SAF profiles defined in the class OPERCMDS. You can define a single profile to represent all VARY TCPIP commands and/or you can specify individual profiles for each VARY TCPIP command option.

The format of the profile name is:

MVS.VARY.TCPIP.*command*

Where:

- ▶ MVS.VARY.TCPIP is a constant
- ▶ *command* is either a double asterisk, *******, meaning all command options, or a specific VARY TCPIP option name, such as OBEYFILE

An important thing to note about the profile name is that it does not specify a z/OS image name or TCP/IP stack name. Therefore, if there is more than one stack on your z/OS image or your SAF database is shared between multiple z/OS systems, granting access to a command enables that command to be performed by a user against any TCP/IP stack in any z/OS system that shares the database.

Protecting VARY TCPIP at the command level

To specify protection at the command level (any option), you specify the generic OPERCMDS profile with a profile name of MVS.VARY.TCPIP.**. Figure 5-1 shows how this is done using RACF commands. Note that you could also use ISPF to do this.

```
SETROPTS GENERIC(OPERCMDS)
RDEFINE OPERCMDS (MVS.VARY.TCPIP.** ) UACC(NONE)
SETROPTS GENERIC(OPERCMDS) REFRESH
SETROPTS RACLIST(OPERCMDS) REFRESH
```

Figure 5-1 Defining generic VARY TCPIP profile to protect command with all options

Note: If the MVS.VARY.TCPIP.** profile is defined, any user that needs to use any VARY TCPIP command must have CONTROL access to this profile, *regardless* of whether they have access to other MVS.VARY.TCPIP.command profiles.

Protecting VARY TCPIP at the command option level

You may decide to protect the VARY TCPIP command at a more granular level, so that you can control who has authority to use the options of the VARY TCPIP command. To protect the command options, you define the particular VARY TCPIP option that you want to protect as the last qualifier in the profile name. Exceptions to this rule are the VARY TCPIP START and VARY TCPIP STOP commands that are protected together with the profile named MVS.VARY.TCPIP.STRTSTOP. See Table 5-1 on page 77 for a list of RACF profile names to protect the VARY TCPIP command options.

Table 5-1 List of RACF profiles to protect various VARY TCPIP console commands

RACF Profile Name	Command Protected
MVS.VARY.TCPIP.**	All VARY TCPIP options
MVS.VARY.TCPIP.DROP	VARY TCPIP,,DROP
MVS.VARY.TCPIP.OBEYFILE	VARY TCPIP,,OBEYFILE
MVS.VARY.TCPIP.PKTTRACE	VARY TCPIP,,PKTTRACE
MVS.VARY.TCPIP.STRTSTOP	VARY TCPIP,,START or VARY TCPIP,,STOP

Figure 5-2 shows the VARY TCPIP,,STOP and VARY TCPIP,,START commands being protected.

```
RDEFINE OPERCMDS MVS.VARY.TCPIP.STRTSTOP UACC(NONE)
SETROPTS RACLIST(OPERCMDS) REFRESH
```

Figure 5-2 Defining specific VARY TCPIP profile to protect VARY TCPIP,,STOP/START commands

SAF checking

The SAF check is performed whenever a z/OS console VARY TCPIP command is attempted. A SAF check is performed against the generic profile first, if it exists. If the generic profile exists, and you do not have CONTROL access to it, you will be denied access to any VARY TCPIP command.

If a profile exists for the specific VARY TCPIP command option that you are attempting, then a check is made against that specific profile, in addition to the generic profile (as explained above). If you do not have CONTROL access to the profile, you will be denied access to the VARY TCPIP command. This is slightly different from other uses of RACF by TCPIP. In this case, access is required only for the most specific matching resource.

5.1.2 VARY TCPIP command security scenario

The command V TCPIP,TCPIPC,STOP,OSA22E0 was chosen to test the console RACF security profiles. The command output is shown in Figure 5-3 before any RACF security profiles were defined to the system. The SAF profile that controls the V TCPIP,,STOP command (in addition to the generic MVS.VARY.TCPIP.** if defined) is MVS.VARY.TCPIP.STRTSTOP.

```
V TCPIP,TCPIPC,STOP,OSA22E0
EZZ0060I PROCESSING COMMAND: VARY TCPIP,TCPIPC,STOP,OSA22E0
EZZ0053I COMMAND VARY STOP COMPLETED SUCCESSFULLY
EZZ4315I DEACTIVATION COMPLETE FOR DEVICE OSA22E0
```

Figure 5-3 VARY TCPIP,TCPIPC,STOP command output. No RACF profiles defined yet.

For the first test, we just defined the generic MVS.VARY.TCPIP.** profile to protect all options of the V TCPIP command. For the second test, we additionally defined the MVS.VARY.TCPIP.STRTSTOP profile to protect the V TCPIP,,STOP command. Both times, unauthorized use of the command caused the expected RACF error.

Define the generic profile to protect all V TCPIP commands

The generic profile MVS.VARY.TCPIP.** was added to the OPERCMDS class with UACC(NONE) as shown in Figure 5-1 on page 76. At this stage, no user had any access to this profile. A user then attempted a V TCPIP,TCPIPC,STOP,OSA22E0 command from the console, which resulted in the RACF error shown in Figure 5-4. Note that the profile being SAF checked was MVS.VARY.TCPIP.**

```
V TCPIP,TCPIPC,STOP,OSA22E0
IEE345I VARY      AUTHORITY INVALID, FAILED BY SECURITY PRODUCT
ICH408I USER(FOCAS  ) GROUP(SYS1  ) NAME(PETER FOCAS IBM NZ  )
MVS.VARY.TCPIP CL(OPERCMD5)
INSUFFICIENT ACCESS AUTHORITY
FROM MVS.VARY.TCPIP.** (G)
ACCESS INTENT(UPDATE ) ACCESS ALLOWED(NONE  )
```

Figure 5-4 RACF error for VARY TCPIP STOP command showing generic profile violation

Define the specific profile to protect the V TCPIP, STOP command

The specific profile MVS.VARY.TCPIP.STRTSTOP was added to the OPERCMDS class with UACC(NONE) as shown in Figure 5-2 on page 77. At this stage, then, both the MVS.VARY.TCPIP.STRTSTOP and the generic MVS.VARY.TCPIP.** profiles were defined, and the user did not have access to either of them.

After the V TCPIP,TCPIPC,STOP,OSA22E0 command was issued, Figure 5-5 shows the profile name that is causing the access problems is MVS.VARY.TCPIP.** even though the MVS.VARY.TCPIP.STRTSTOP profile is defined. This confirms what was said in “Protecting VARY TCPIP at the command level” on page 76, that the generic profile is always checked first, if defined.

```
IEE345I VARY      AUTHORITY INVALID, FAILED BY SECURITY PRODUCT
ICH408I USER(FOCAS  ) GROUP(SYS1  ) NAME(PETER FOCAS IBM NZ  )
MVS.VARY.TCPIP CL(OPERCMD5)
INSUFFICIENT ACCESS AUTHORITY
FROM MVS.VARY.TCPIP.** (G)
ACCESS INTENT(UPDATE ) ACCESS ALLOWED(NONE  )
```

Figure 5-5 RACF error for VARY TCPIP, STOP command shows generic profile name

If the generic profile MVS.VARY.TCPIP.** is defined, any user who wants to enter a VARY TCPIP command must have CONTROL access to it. The user was therefore given CONTROL access and entered the command V TCPIP,TCPIPC,STOP,OSA22E0 again.

Now that we have CONTROL access to the generic profile MVS.VARY.TCPIP.**, the SAF check is for the profile that controls the VARY TCPIP, STOP command, which is MVS.VARY.TCPIP.STRTSTOP. Figure 5-6 on page 79 shows the RACF error resulting when the user does not have CONTROL access to the specific profile.

Once the user was given CONTROL access to both the generic profile MVS.VARY.TCPIP.** and the specific profile MVS.VARY.TCPIP.STRTSTOP, the V TCPIP,TCPIPC,STOP,OSA22E0 command returned the expected results.

```

V TCPIP,TCPIPC,STOP,OSA22E0
EZZ0060I PROCESSING COMMAND: VARY TCPIP,TCPIPC,STOP,OSA22E0
IEE345I VARY      AUTHORITY INVALID, FAILED BY SECURITY PRODUCT
ICH408I USER(FOCAS  ) GROUP(SYS1  ) NAME(PETER FOCAS IBM NZ  )
  MVS.VARY.TCPIP.STRTSTOP CL(OPERCMD5)
  INSUFFICIENT ACCESS AUTHORITY
  ACCESS INTENT(CONTROL) ACCESS ALLOWED(NONE  )
EZZ0059I MVS.VARY.TCPIP.STRTSTOP COMMAND FAILED: NOT AUTHORIZED

```

Figure 5-6 RACF error from unauthorized use of TSO NETSTAT command - specific profile

5.2 TSO NETSTAT and UNIX onetstat command security

The TSO NETSTAT command, and the UNIX shell `onetstat` command can be protected from unauthorized use at both the command level (NETSTAT with any option) and the command option level. By defining a SAF profile to represent the NETSTAT command and option, you can grant permission by user or group to the NETSTAT command and/or its options.

5.2.1 RACF profile details

The SERVAUTH class is used to define a profile to protect the NETSTAT command. The format of the profile name is:

EZB.NETSTAT.sysname.tcprocname.option

Where:

- ▶ EZB.NETSTAT is constant
- ▶ sysname is the z/OS image name
- ▶ cprocname is the TCP/IP stack name
- ▶ option is either an asterisk, '*', meaning all options, or a specific NETSTAT option name

As mentioned, you can protect NETSTAT/`onetstat` at the command level and/or the command option level.

Protecting NETSTAT/onetstat at the command level

To specify protection at the command level, you specify the SERVAUTH profile with a command option of an asterisk, "*", and you define the SERVAUTH profile to be generic. Figure 5-7 shows how this is done using RACF commands, assuming the system name is SC63 and the TCP/IP stack name is TCPIPC.

```

SETROPTS GENERIC(SERVAUTH)
RDEFINE SERVAUTH (EZB.NETSTAT.SC63.TCPIPC.*) UACC(NONE)
SETROPTS GENERIC(SERVAUTH) REFRESH
SETROPTS RACLIST(SERVAUTH) REFRESH

```

Figure 5-7 Defining generic NETSTAT profile to protect the command with all options

Note that the SETROPTS GENERIC(SERVAUTH) needs to be done only once.

Protecting NETSTAT/onetstat at the command option level

You may decide to protect the NETSTAT/**onetstat** command at a more granular level, so that you can control who has authority to use the options of the NETSTAT/**onetstat** command. To protect the command options, you define the particular NETSTAT option that you want to protect as the last qualifier in the profile name. Figure 5-8 shows the NETSTAT HOME or **onetstat -h** command being protected for TCP/IP system TCPIPC on z/OS system SC63.

```
RDEFINE SERVAUTH (EZB.NETSTAT.SC63.TCPIPC.HOME) UACC(NONE)
SETROPTS RACLIST(SERVAUTH) REFRESH
```

Figure 5-8 Defining NETSTAT profile to protect NETSTAT/onetstat command with home option

Restriction: The NETSTAT DROP command is internally implemented as a z/OS console command VARY TCPIP,,DROP, and as such is protected by the SAF profile MVS.VARY.TCPIP.DROP, not by a NETSTAT SAF profile.

SAF checking

The NETSTAT SAF check is performed whenever a TSO NETSTAT or UNIX **onetstat** command is attempted. A SAF check is performed against the most specific profile name first. If a profile for the specific command option doesn't exist, then a check is made against the generic profile (the profile with a command option specified as "**").

5.2.2 NETSTAT security scenario

Our system name at ITSO is SC63 and the TCP/IP stack name is TCPIPC. Our tests were to use the TSO NETSTAT HOME command. For the first test, we just defined the generic NETSTAT profile to protect all options of the NETSTAT command. For the second test, we defined the profile to protect the NETSTAT HOME command. Both times, unauthorized use of the command caused the expected RACF error.

In the discussions that follow, wherever the TSO NETSTAT command is mentioned, the OMVS **onetstat** command is also implied.

Define the NETSTAT generic profile to protect all NETSTAT commands

The generic profile EZB.NETSTAT.SC63.TCPIPC.* was added to the SERVAUTH class with UACC(NONE) as shown in Figure 5-7 on page 79. At this stage, no user had any access to this profile. A user then attempted a TSO NETSTAT HOME command, which resulted in the RACF error shown in Figure 5-9. Note that the profile being SAF checked was EZB.NETSTAT.SC63.TCPIPC.HOME, but since that did not exist, the profile EZB.NETSTAT.SC63.TCPIPC.* was checked. This latter profile was the generic profile that failed the SAF check.

```
ICH408I USER(FOCAS ) GROUP(SYS1 ) NAME(PETER FOCAS IBM NZ ) 25
EZB.NETSTAT.SC63.TCPIPC.HOME CL(SERVAUTH)
INSUFFICIENT ACCESS AUTHORITY
FROM EZB.NETSTAT.SC63.TCPIPC.* (G)
ACCESS INTENT(READ ) ACCESS ALLOWED(NONE ) ***
```

Figure 5-9 RACF error from unauthorized use of TSO NETSTAT command - generic profile

Define the NETSTAT profile to protect the NETSTAT HOME command

The specific profile EZB.NETSTAT.SC63.TCPIPC.HOME was added to the SERVAUTH class with UACC(NONE) as shown in Figure 5-8 on page 80. At this stage, no user had any access to this profile. A user then attempted a TSO NETSTAT HOME command, which resulted in the RACF error shown in Figure 5-10. Note that the profile that has been SAF checked is now EZB.NETSTAT.SC63.TCPIPC.HOME rather than EZB.NETSTAT.SC63.TCPIPC.

```
ICH408I USER(FOCAS ) GROUP(SYS1 ) NAME(PETER FOCAS IBM NZ )
EZB.NETSTAT.SC63.TCPIPC.HOME CL(SERVAUTH)
INSUFFICIENT ACCESS AUTHORITY
ACCESS INTENT(READ ) ACCESS ALLOWED(NONE )
Access to Netstat HOME denied - SAF RC is 00000008
***
```

Figure 5-10 RACF error from unauthorized use of TSO NETSTAT command - specific profile

The OMVS command equivalent of TSO NETSTAT HOME is **onetstat -h**. The same user that attempted the TSO NETSTAT command in Figure 5-10 used the command **onetstat -h -p tcpipc** (the -p parameter targets the TCP/IP stack named TCPIPC) and got the expected error as shown in Figure 5-11.

```
FOCAS @ SC63: />onetstat -h -p tcpipc
EZZ2385I Access to Netstat -h denied - SAF RC is 00000008
FOCAS @ SC63: />
```

Figure 5-11 OMVS error from unauthorized use of the onetstat command

5.2.3 Further reading

For more information on all the profile names used to protect the various NETSTAT command options, see *z/OS V1R2.0 CS: IP System Administrator's Commands*, SC31-8781.



Part 3

Network security

This part of the redbook covers the use of IPSec to secure network traffic. IPSec can be used for authentication, encryption, or both. In order to use IPSec, you must implement the z/OS Firewall supplied in the z/OS Secureway Security server (RACF) component.

The chapters in this part provide a brief overview of the z/OS Firewall and IPSec, followed by a fairly in-depth example of creating a virtual private network (VPN) using the z/OS Firewall and the firewall configuration GUI client.

An introduction to firewalls in general and the z/OS Firewall is discussed in Chapter 6, “Firewall concepts” on page 85.

An introduction to the IPSec protocol and Internet Key Exchange (IKE), is presented in Chapter 7, “IPSec and virtual private networks (VPN)” on page 91.

Finally, using the z/OS Firewall for implementing a VPN with IPSec is discussed in Chapter 8, “Implementing IPSec with z/OS Firewall Technologies” on page 99.

Firewall concepts

A firewall is a computer used to separate a secure network from a non-secure network (see Figure 6-1). Such networks are typically based on the TCP/IP protocol, but the concept of a firewall concept is not restricted to just TCP/IP.

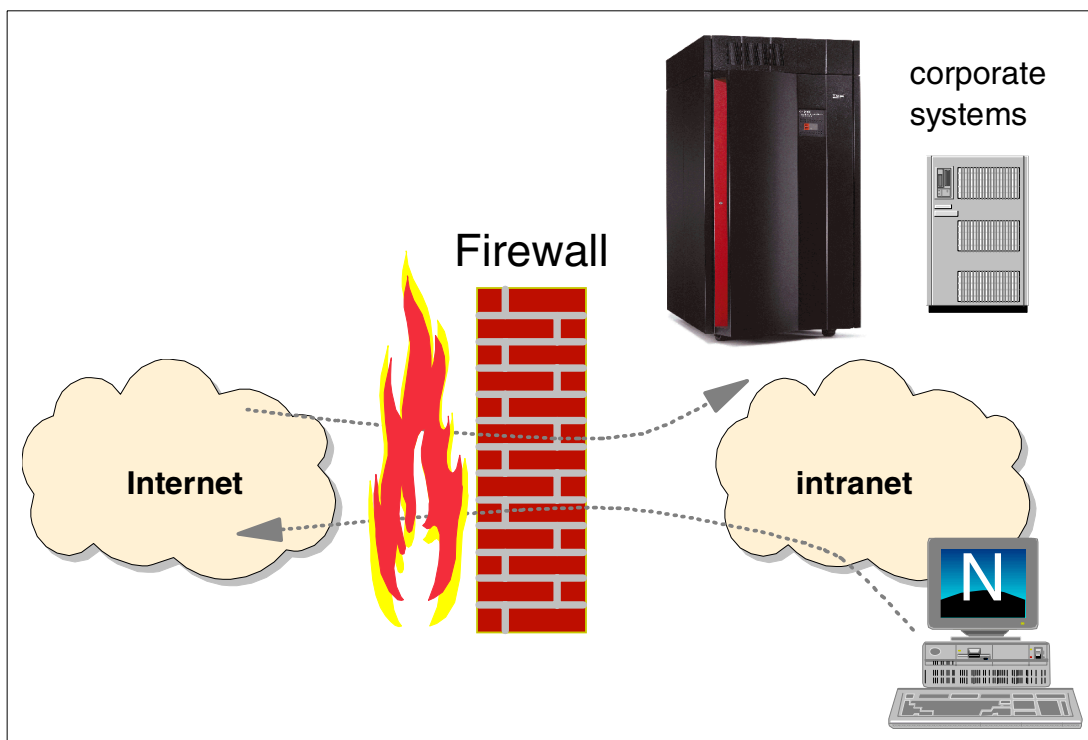


Figure 6-1 The firewall concept

Firewalls have become an important concept in TCP/IP-based networks, because the global Internet is a TCP/IP-based network and is often perceived as being a non-secure place to enter or traverse. Yet you still want your intranet (perceived as being a secure place) to be connected to the non-secure Internet.

The reasons for establishing connections between an intranet and the Internet are many, but generally fall into two categories:

- ▶ You want to provide a service to the Internet community or want to conduct business on the Internet.
- ▶ You want to allow your internal employees to access the vast amount of services on the Internet, as well as the ability to exchange or share information with other users on the Internet or through the Internet.

At this point, it might be useful to define the following terms:

- ▶ The term *intranet* refers to an internal TCP/IP network.
- ▶ The term *Internet* refers to the worldwide interconnected network.
- ▶ The term *extranet* refers to TCP/IP networks of different companies connected with a secure connection, perhaps using virtual private network technology (VPN).

Doing e-business on the Internet is very different from just serving static information out of a Web server. e-business means that you have to establish an environment where users on the Internet are able to interact with the applications and data that your daily existence as a company is based on and relies upon.

That data and those applications are likely, to a large extent, to be located in your z/OS environment, which means that you probably already are, or in the near-term future will be, challenged with the request to establish Internet access to your z/OS production environment.

When you connect your intranet to the Internet and define a strategy for how your firewall should function, you may think that it is sufficient to block all types of traffic that represent a risk, and allow the remaining traffic to pass through the firewall. However, such a strategy is based on the assumption that all risks are known in advance and that existing well-behaving traffic will remain well-behaving. Such an assumption is a mistake. New ways of exploiting existing applications and well-known application protocols are being found every week, so an application that may be considered harmless today may be the instrument of an attack tomorrow.

6.1 General guidelines for implementing firewalls

A few general guidelines for implementing firewall technologies are worth including in this context.

Before you start connecting your intranet to the Internet, define a security policy for how your firewall should function and how demilitarized zones should be configured. Decide what type of traffic is allowed through the firewall and under what conditions. Decide what kind of servers are to be placed in demilitarized zones and what type of traffic is allowed between the demilitarized zone and the intranet.

When actually configuring your firewall, start by disallowing everything and then proceed by enabling those services you have defined in your security policy. Everything that is not specifically allowed should be prohibited.

If you establish more than a single gateway between your internal network and the Internet, make sure that all gateways implement the same level of security. It is common practice to use different firewall products in a vertical setup (product A between the Internet and the demilitarized zone and product B between the demilitarized zone and the intranet). That way, a malicious hacker exploiting a vulnerability in product A is still stopped by product B. Of course, it does not make sense to use this concept in a horizontal setup (one gateway uses product A, the other one product B) because a malicious hacker will get in at the weakest link.

If you build a perfect firewall on one end of your network while users on the other end dial in to the Internet from their LAN-attached PCs, enabling those PCs to act as IP routers between your internal network and the Internet, a malicious hacker is soon going to exploit that back door into your network instead of wasting his time trying to break through your firewall.

One of the most important aspects of a firewall is its ability to log both successful and rejected access events. However, these logs are worth nothing if you do not set up daily administrative procedures to analyze and react to the information that can be derived from these logs.

By analyzing the firewall logs, you should be able to detect if unauthorized accesses were attempted and if your firewall protection succeeded in rejecting such attacks, or if it failed and allowed an intruder to gain access to resources that should not have been accessed. In addition, it might be a good idea to install an intrusion-detection system.

This list is not all-inclusive, but merely points out some of the most important aspects of implementing firewall technologies in your network.

So far, we have only considered the Internet to be the non-secure place, while your internal network has been considered the secure place. However, that may in some situations be an oversimplification. For example, consider a research department that works with highly confidential information. In such an environment, you may want to protect that research department from your regular users by implementing a firewall between your regular internal network and the network in your research department.

6.2 Firewall categories

There are many firewall technologies available, but they can in general be grouped into two major categories:

- ▶ Those that allow IP packets to be routed between two or more networks, namely packet-filtering routers.
- ▶ Those that disable IP routing, but relay data through specialized application programs, namely application-level gateways or proxies.

6.2.1 Packet filtering

A packet-filtering router, as shown in Figure 6-2 on page 88, is a special type of IP router. What differentiates a firewall packet-filtering router from a normal IP router is that it applies one or more technologies to analyze the IP packets and decide if a packet is allowed to flow through the firewall or not. Such a firewall is sometimes also referred to as a screening filter, or router firewall.

Depending on the type of packet, some packet-filtering techniques only act on data in the headers of individual packets, while others also look at data. The traditional packet-filtering router is stateless (each packet is handled independently), but there are products that save state over multiple packages and base their actions on the state information.

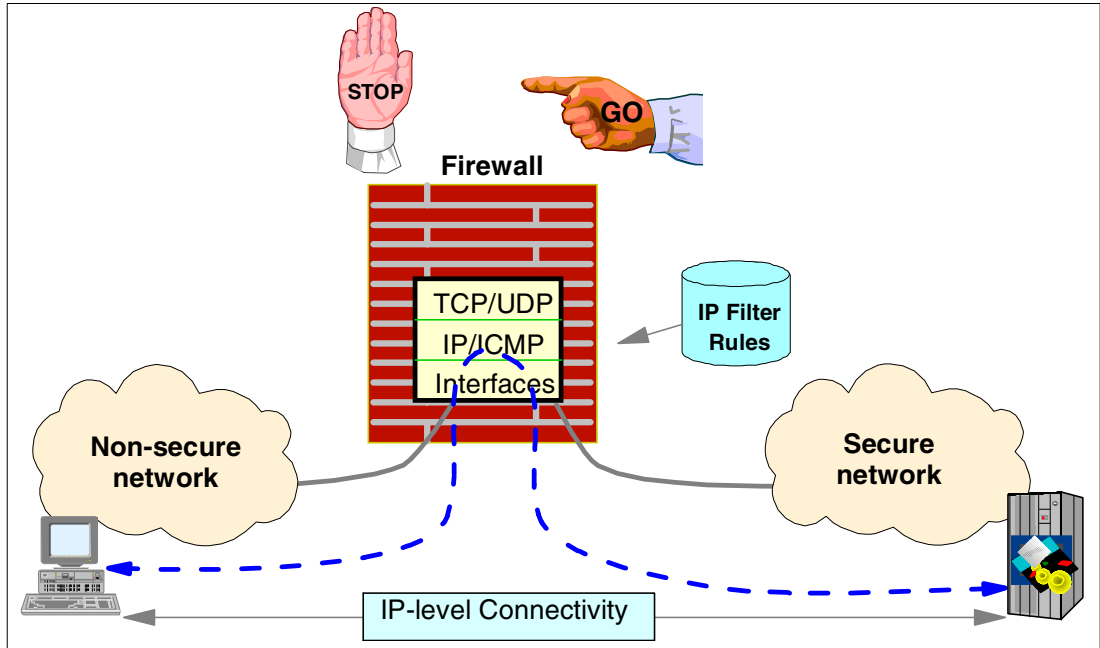


Figure 6-2 Packet-filtering firewall

6.2.2 Application-level gateway

An application-level gateway, sometimes referred to as a bastion host, is a machine that disables IP-level routing between the non-secure network and the secure network, but allows specialized application gateway programs (termed proxies) that run on the firewall to communicate with both the secure network and the non-secure network. See Figure 6-3.

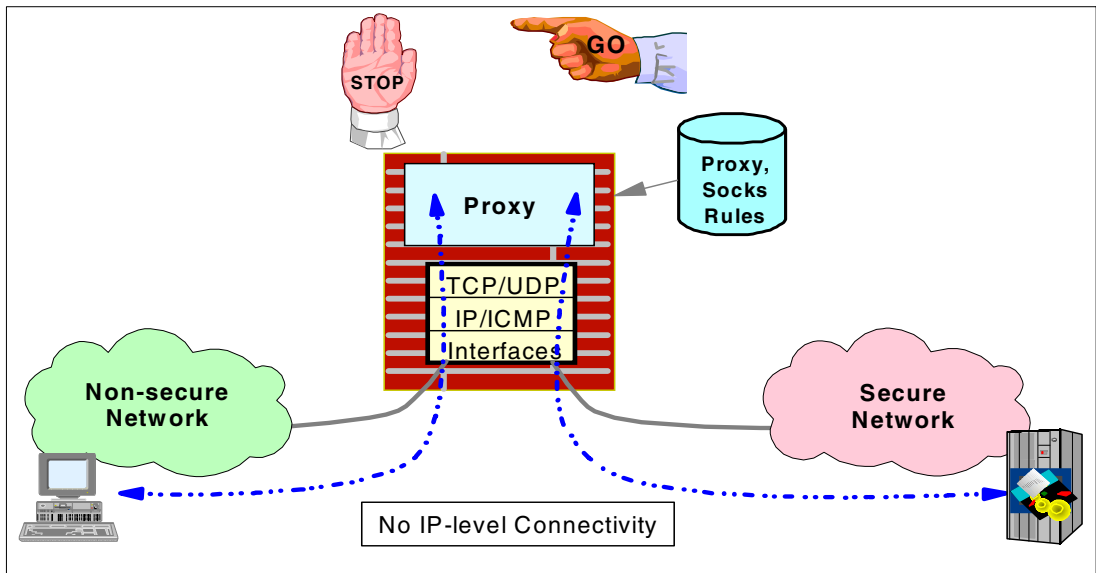


Figure 6-3 Application gateway firewall

The proxy applications on the firewall act as relay applications between users or applications on the secure and the non-secure networks. Examples of such proxy applications are HTTP or FTP proxy servers. The SOCKS server is also an application-level gateway, but a special kind, sometimes referred to as a circuit level gateway (at the transport layer). A SOCKS server can relay all TCP connections and UDP datagrams, not just HTTP or FTP sessions. It does not provide any extra packet processing or filtering, and unlike proxy servers, it is often used for outbound connections z/OS through a firewall.

A firewall may not always have to be configured as either a packet-filtering router or as a proxy. It may be configured to act as a packet-filtering router for certain application protocols, while it acts as a proxy for other applications.

An excellent discussion of firewall technologies can be found in the redbook *TCP/IP Tutorial and Technical Overview*, GG24-3376.

6.3 z/OS Firewall Technologies

Starting with OS/390 V2R4, firewall functions have been made available on z/OS. The following functions are supported:

- ▶ IP filtering
- ▶ Network Address Translation (NAT)
- ▶ Virtual private networks (VPN)
- ▶ FTP proxy server
- ▶ SOCKS server
- ▶ Domain name services

We would suggest using the z/OS Firewall Technologies component to protect a z/OS server machine from the network(s) it is connected to, rather than as a firewall that handles all traffic between the Internet and an intranet. For more information on z/OS Firewall Technologies, see *z/OS SecureWay Security Server Firewall Technologies*, SC24-5922.

6.4 The demilitarized zone

The demilitarized zone (DMZ) is a term often used when describing firewall configurations. Figure 6-4 on page 90 shows a typical example. A DMZ is an isolated subnet between your secure network and the Internet. Much as the no-man's land between two entrenched armies, anyone can enter it, but the only things present are those that you wish to allow access to anyway. Nowadays, we don't plant mines in a demilitarized zone; instead, it is an area in which you place the Web servers and other servers for public access, but which you also wish to protect to some degree.

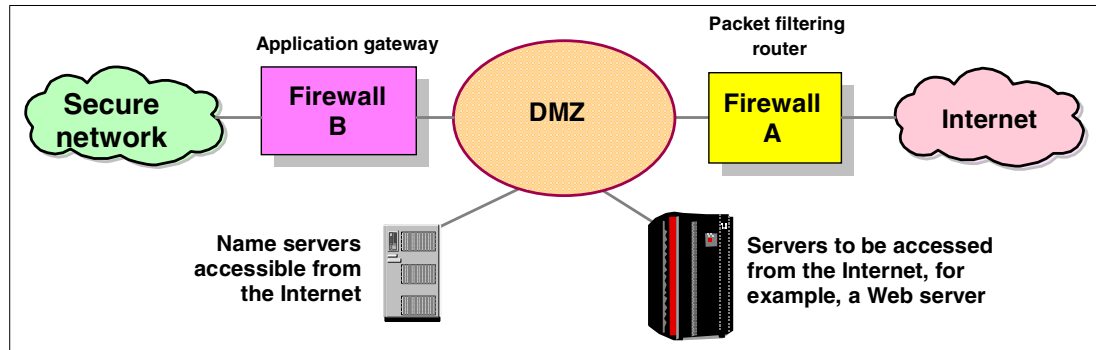


Figure 6-4 A demilitarized zone

This is achieved by placing an outer firewall (often a packet-filtering router) between the Internet and the servers in the DMZ, and another firewall (often an application-level gateway) between your secure network and the DMZ. The outer firewall is designed to allow into the DMZ only those requests you wish to receive at your Web servers, but could also be configured to block denial-of-service attacks and to perform network address translation of the servers in your DMZ. The inner firewall is designed to prevent unauthorized access to your secure network from the DMZ and also perhaps to prevent unauthorized access from your secure network to the DMZ or the connected non-secure network. IPSec authentication can be used as an alternative means to control access to secure network.

When you put a z/OS server into a DMZ, we would strongly suggest that you use z/OS Firewall Technologies. You should use z/OS Firewall Technologies to block all traffic into and out of your z/OS server that does not belong to the services you are going to offer from this server. This control should be in place even if you already have a filtering router or firewall between the insecure network and this server.

IPSec and virtual private networks (VPN)

A virtual private network (VPN) provides secure connections across the Internet, by establishing a “tunnel” between two secure networks. It is a generic solution that is both application and protocol independent. A VPN encapsulates the IP datagram into another IP datagram in order to maintain data privacy. It can be used by two disparate parts of a corporation to connect their internal private networks by means of a non-secure network such as the Internet. An example of a VPN configuration is shown in Figure 7-1.

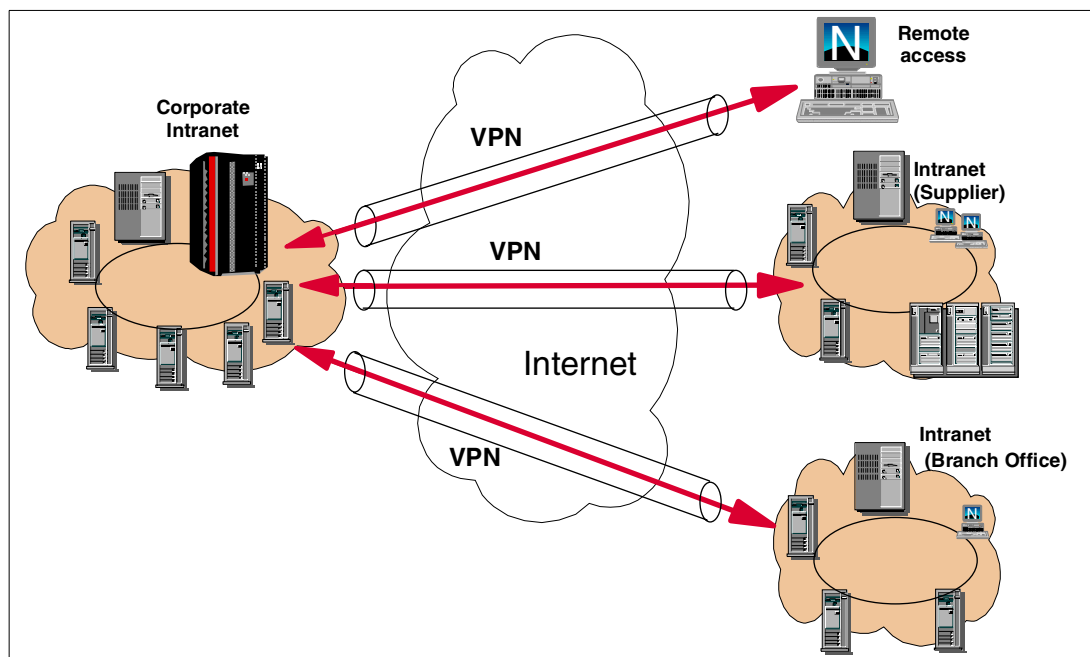


Figure 7-1 Virtual private networks

7.1 IPSec

Figure 7-2 shows the TCP/IP layered protocol stack with the security-related protocols associated with each layer:

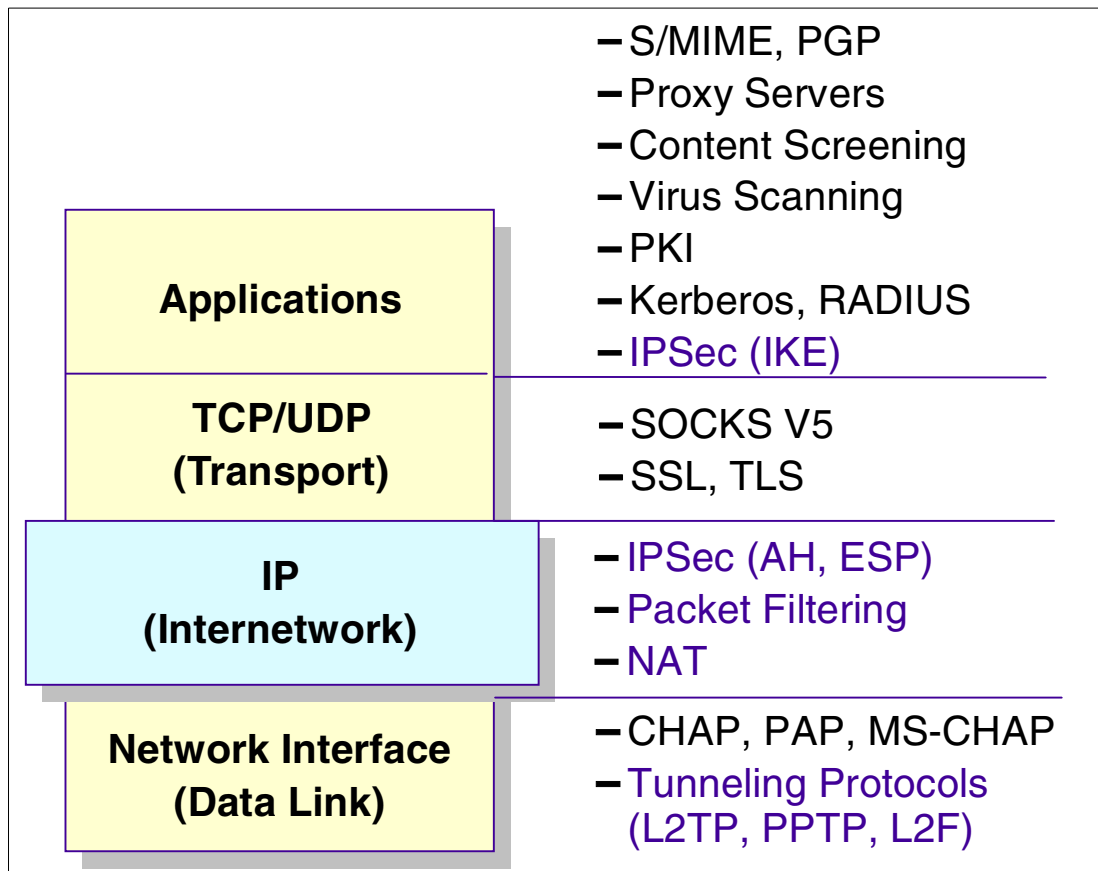


Figure 7-2 The TCP/IP protocol stack and security-related protocols

Within the layered communications protocol stack model, the network layer (IP in the case of the TCP/IP stack) is the lowest layer that can provide end-to-end security. Network-layer security protocols provide blanket protection for all upper-layer application data carried in the payload of an IP datagram, without requiring a user to modify the applications.

The IP Security Architecture (IPSec) open framework is defined by the IPSec Working Group of the IETF. IPSec is called a framework because it provides a stable, long-lasting base for providing network-layer security. It can accommodate today's cryptographic algorithms, and can also accommodate newer, more powerful algorithms as they become available. IPv6 implementations must support IPSec, and IPv4 implementations are strongly recommended to do so.

IPSec is comprised of a number of components described in individual RFCs that are designed to operate together:

- ▶ Security Protocols - IP Authentication Header (AH) provides data origin authentication, data integrity, and replay protection, while IP Encapsulating Security Payload (ESP) provides data confidentiality, data origin authentication, data integrity, and replay protection.

- ▶ Security Associations - an SA is a kind of session between two hosts defining the protocols to be used when transmitting data. ISAKMP (Internet Security Association and Key Management Protocol) is a generic framework for negotiating SAs and keys.
- ▶ Key Management - Internet Key Exchange (IKE) provides a method for automatically setting up security associations and managing and exchanging their cryptographic keys.

7.1.1 Security Associations

An IPSec Security Association (SA) defines the set of protocols and, with these, the negotiated algorithms and keys that are to be used when transmitting data between two hosts. An SA for data traffic is always unidirectional, so for a pair of hosts that are to communicate securely, at least two SAs, one for each direction, are needed. This differs from other protocols that make use of sessions, for example SSL. An SSL session covers the transmission in both directions.

A Security Association has three parts:

1. A Security Parameter Index (SPI), a unique 32-bit value identifying the SA
2. The source IP address
3. The protocol used (AH or ESP)

Negotiating Security Associations (ISAKMP and IKE)

Before any data can be sent between two hosts using IPSec, an SA needs to be established. The IPSec architecture provides two methods for establishing an SA: a manual tunnel or ISAKMP/IKE.

With manual tunnels, the SA and cryptographic keys are generated on one of the hosts (the tunnel owner), transferred to the other host (the tunnel partner) via an out-of-band transport mechanism, and then imported. This procedure needs to be repeated whenever the validity of the keys has expired and new cryptographic keys need to be generated.

Contrary to manual tunnels, ISAKMP and IKE provide automatic management of sessions and keys. ISAKMP provides a generic framework for the negotiation of SAs and keying material. It defines the procedures and packet formats to establish, negotiate, modify, and delete SAs, but it does not provide any specific key-generation techniques or cryptographic algorithms.

Internet Key Exchange (IKE) is based on two protocols: Oakley (see *The Oakley Key Determination Protocol*, by H. Orman; RFC 2412, November 1998) and SKEME (see *SKEME: A Versatile Secure Key Exchange Mechanism for the Internet*, by H. Krawczyk; IEEE Proceedings, 1996). For the key exchange, Diffie-Hellman (DH) shares are used and the shared key thus obtained is used to derive the keys for data encryption and message authentication. Authentication can be performed with one of three alternatives:

- ▶ Digital signatures
- ▶ Public key encryption
- ▶ A shared secret (a key previously known to both parties)

The use of DH shares causes the connection to have a property called “perfect forward secrecy”. This means that even if the keys for one session are completely compromised, the keys for previous sessions are still safe.

Phases: it takes two

Two hosts can communicate with each other in many different ways that may need different sorts of protection. For instance, some traffic may need encryption and authentication while other traffic may only need authentication.

IKE uses a two-phase approach to be able to meet these different needs with minimal overhead. In phase 1, an ISAKMP SA is negotiated to create a secure, authenticated channel between the two hosts. The ISAKMP SA is a single, bi-directional Security Association. In phase 2, the SAs for the individual type of traffic (one SA for each direction) are negotiated using the authenticated channel established in phase 1.

Due to the Diffie-Hellman key exchange and the authentication, phase 1 is computationally rather expensive. Phase 2 does not involve key exchange nor authentication and is much less expensive. Performing phase 1 just once for a pair of hosts and then multiple phase 2 operations for the individual connections is a concept that can improve performance considerably.

Identity protection

In phase 1, certificates and authentication data are exchanged between the hosts. IKE offers *identity protection*, meaning that all information that could identify a host to an attacker or eavesdropper can be encrypted. Depending on whether identity protection is really required, IKE supports two modes for phase 1: *main mode* offers identity protection while *aggressive mode* does not. In main mode, a shared, secret key is established before the identification information (for instance, the host's digital certificate) is sent. For a diagram showing IKE main mode, see Figure 7-3.

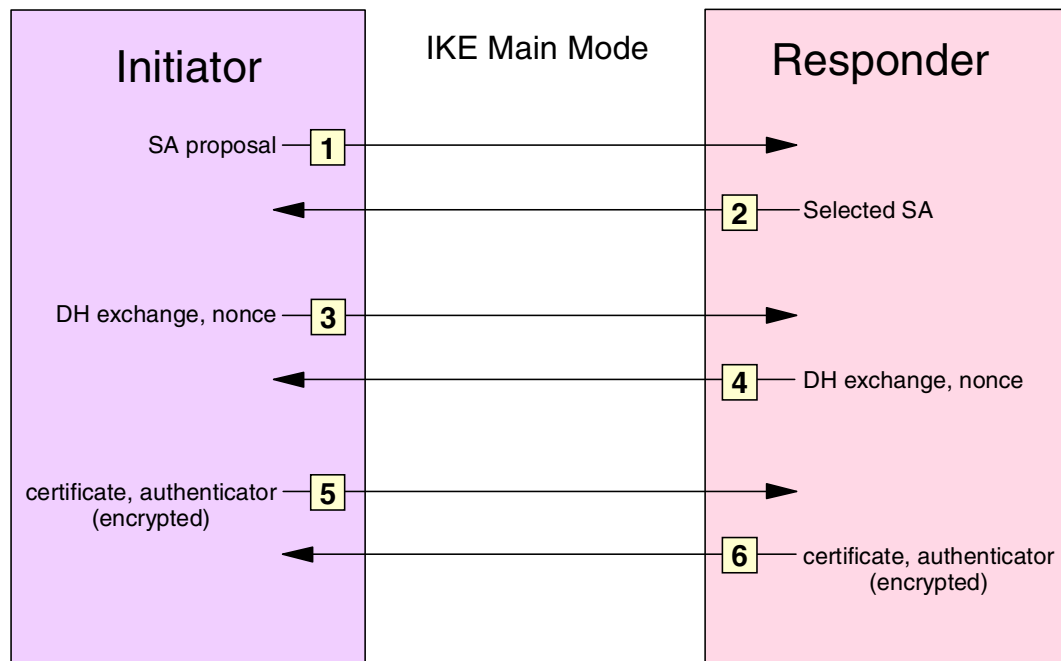


Figure 7-3 IKE phase 1: main mode

Aggressive mode does not require the DH key exchange to be completed before sending the remaining information. Therefore, there is only one exchange of messages in aggressive mode (see Figure 7-4 on page 95).

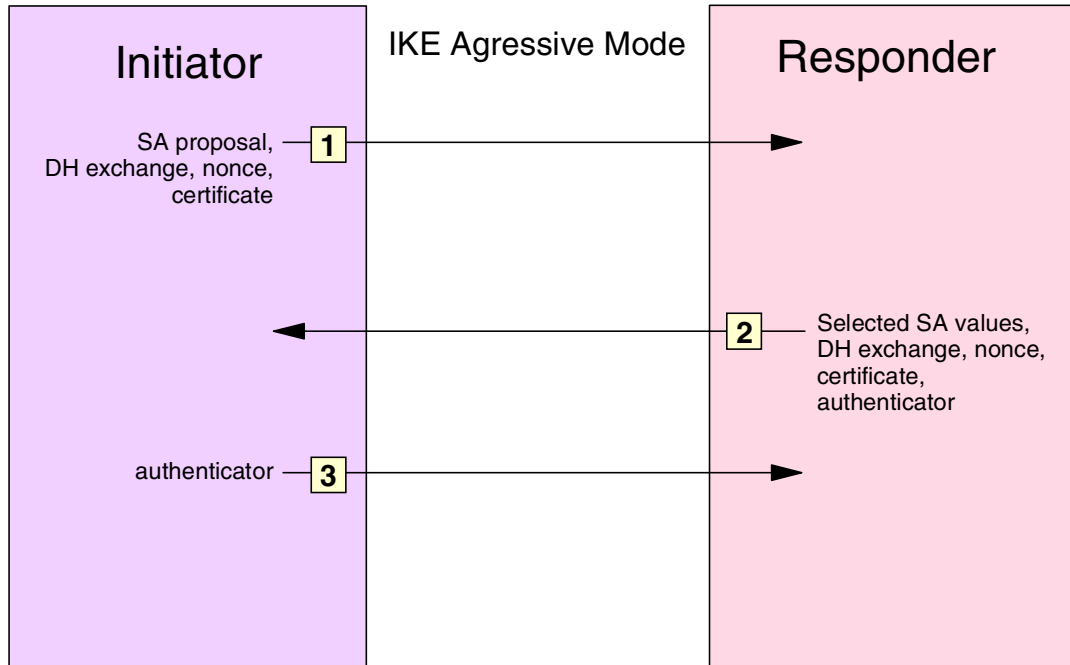


Figure 7-4 IKE phase 1: aggressive mode

The exchange of messages taking part in phase 2 (negotiation of the SAs for the individual type of traffic) is called *quick mode*. In this mode, the pair of SAs for the intended type of communication is set up. The required keys for encryption and message authentication are generated from the shared key obtained in phase 1.

7.1.2 Transmitting data with IPSec

When a host wants to transmit one or more packets to another host it had not contacted before, it will perform the necessary IKE exchanges to set up the required SAs with the other hosts. Once this has all been performed and the necessary keys are generated, the host proceeds with sending the first packet.

IPSec has two formats for sending data that are serving slightly different purposes. *Authentication Header (AH)* provides for message authentication and replay protection, whereas *Encapsulating Security Payload (ESP)* provides for data encryption in addition to message authentication and replay protection. The SA for a communication selects whether AH, ESP, or a combination of both is to be used.

Depending on the type of VPN connection between the two hosts, there are two modes, *tunnel mode* and *transport mode*, that are to be used.

ESP and AH in transport mode

If a VPN connection is being established between two hosts that are the endpoints for the packets transmitted between them, transport mode should be used. Figure 7-5 on page 96 shows the format of Authentication Header (AH) in transport mode.

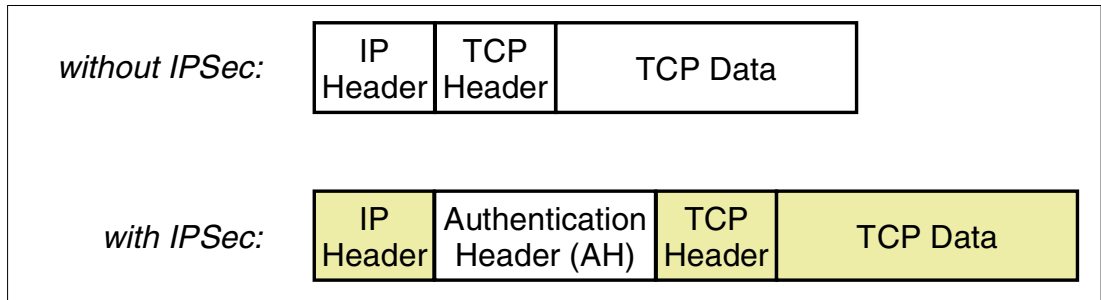


Figure 7-5 AH in transport mode

The message authentication applied by AH protects the parts of the packet that are shaded in Figure 7-5. Note that although the IP header is shaded in the diagram, parts of it are not authenticated because they can change in transit between sender and receiver.

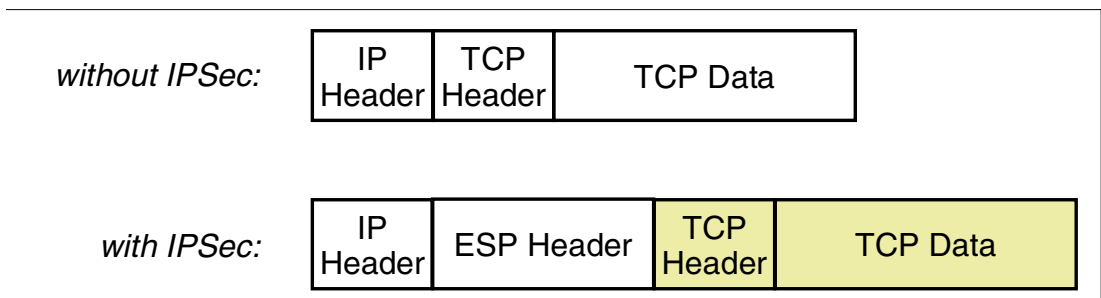


Figure 7-6 ESP in transport mode

With the Encapsulating Security Payload (ESP) format in transport mode, the TCP header and data are encrypted and, optionally, authenticated. But as can be seen in Figure 7-6 (where the protected areas are shaded), the IP header is afforded no protection at all. However, this should not be a problem because sending and receiving hosts have been authenticated and verified in the SA.

ESP in tunnel mode

A common application of VPNs is the use of a protected tunnel between two secure networks. IPSec-capable firewalls at each end of the tunnel encrypt the packets they send from the secure network through the tunnel; they decrypt the packets they receive from the tunnel and route them to the destination hosts. In this scenario, the SAs do not authenticate the destination hosts (just the firewalls) and an attacker's modification of the IP headers could go undetected.

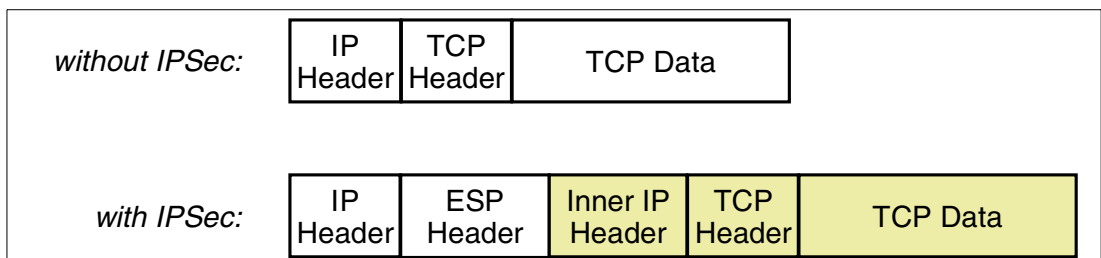


Figure 7-7 ESP in tunnel mode

In this environment, tunnel mode is to be used. Figure 7-7 on page 96 shows the format of ESP packets in this mode; again, protected areas are shaded. The complete original packet, including the original IP header, is used as payload for an ESP packet. The inner IP header has the address of the destination host, while the outer IP header addresses the firewall at the end of the tunnel. In this way, the complete packet including the IP header is protected.

In some cases, the AH and ESP formats are combined (applied one after the other) in order to reap both the benefits of IP header authentication with AH and payload (data) encryption with ESP.



Implementing IPsec with z/OS Firewall Technologies

The word *firewall* was originally used in building and construction to denote the heavy walls put in place to prevent the spreading of fires. In networking, it is now used for a process in a computer network that protects resources within its scope against unauthorized usage.

Firewalls become more and more important as a way to protect information technology assets from being abused. This abuse may be willful destruction of important information or simply unauthorized use of resources. With z/OS Firewall Technologies you may erect a security curtain around your network valuables.

Note: This chapter uses examples from OS/390 V2R10. The Firewall Technologies component of z/OS V1R2 is similar with a few enhancements. The enhancements are described in detail in *z/OS SecureWay Security Server Firewall Technologies, SC24-5922*.

8.1 Introduction

A firewall is a solution consisting of hardware and software working together to limit access to selected resources and to ensure privacy. The z/OS Firewall Technologies component offers a broad spectrum of possibilities on the mainframe to protect network resources.

z/OS Firewall Technologies provides for:

- ▶ Application gateways (proxies)
 - FTP proxy
- ▶ Transparent gateways (SOCKS)
 - SOCKS V4 daemon
- ▶ Packet filtering
 - Filter rules
 - IPSec, virtual private network (VPN) or tunneling
 - zSeries hardware cryptographic facility is used if available
- ▶ Network Address Translation (NAT)
 - Administrator-defined address mapping
 - Address translation in IP headers only
- ▶ Logging
 - Enhanced syslog daemon
 - SMF records
- ▶ Configuration and administration
 - Compatible with AIX, Windows NT and OS/2 (command line and GUI)
 - Commands are valid in OE only, not from TSO
 - Commands create intermediate files, which are used to update the online configuration
 - The z/OS Security Server (RACF) is used to control authorization to maintain network security profiles, in line with general security concepts on mainframes

The above features may be used in any combination.

8.2 Firewall enhancements

z/OS Firewall Technologies offers the following important functions:

- ▶ IP Security
- ▶ Configuration Client GUI

The product can be administered through a Java-based interface that runs on AIX, Windows 95, and Windows NT platforms. It allows you to perform remote configuration and administration in a secure way using RACF authentication and SSL (Secure Sockets Layer) transport.

- ▶ Support of multiple stacks

z/OS Firewall Technologies supports up to eight TCP/IP stacks.

8.3 Installation planning

Firewall Technologies has been part of z/OS (OS/390) since V2R7.

- ▶ Included with the z/OS Security Server are the following:
 - Proxy FTP support
 - SOCKS server (daemon) support
 - Enhanced logging
 - Command-line configuration/administration
 - GUI configuration/administration
- ▶ Included with Communications Server for z/OS are:
 - Network Address Translation (NAT)
 - IP packet filtering
 - IP tunnels (IPSec, or virtual private network)

8.4 Installation, configuration and operation

The following two books are available for planning, installation and reference:

- ▶ *z/OS SecureWay Security Server Firewall Technologies, SC24-5922*

This book gives detailed information on how to get a firewall up and running. Part of it is set up as a cookbook.
- ▶ *Stay Cool on OS/390: Installing Firewall Technology, SG24-2046*

This book provides a wealth of information on:

 - TCP/IP protocol layers
 - Socket types
 - General network security
 - Firewalls
 - Domain name server (DNS)
 - Secure name server
 - IP filter configurations

8.5 Interoperability considerations

z/OS Firewall Technologies support is able to interoperate with any system that supports the IPSec protocol standards described in RFCs 1825 through 1829 or the RFCs 2401-2406 and 2410. The ability to interoperate with another IBM s/390 system or the IBM AIX Firewall is achieved by using the `fwtnn1` (or through the GUI equivalent window) command to export the tunnel definition from one system and then import it to the other. However, the export and import output files are release dependent, so it's necessary to check their format before using them. To interoperate with a non-IBM system, the task requires additional manual configuration. Currently there are two choices:

- ▶ Define and export the tunnel on z/OS

At this point someone must examine the exported files, understand the format of the data, and manually input this data into the non-IBM system using whatever mechanism is provided. In this scenario, the "tunnel owner" refers to the z/OS system and "tunnel partner" refers to the remote system.

- ▶ Define the tunnel on z/OS, and the non-IBM system and determine the values of all parameters that define the tunnel.

At this point someone must manually create two files on the z/OS system in the format of those built during a tunnel export request. These files must be updated with all of the tunnel definition parameters and then imported using the **fwtunn1** command. In this scenario the "tunnel owner" refers to the remote system and "tunnel partner" refers to the z/OS system.

Only after performing one of these steps does the same security association exist on both systems.

An example of how to change an export file to match a different environment is shown, for our test scenario, in 8.8.7, "FWTUNNL export file conversion from z/OS and AIX" on page 118.

Note: For a description of how to define tunnels on various platforms, see *A Comprehensive Guide to Virtual Private Networks*, SG24-5201.

8.6 Sample configuration files

Configuration sample files are shipped with the product.

In `/usr/lpp/fw/etc` you can find:

- ▶ `syslog.conf` - logging server configuration file
- ▶ `fwftp.data` - FTP proxy configuration file
- ▶ `fwftp.deniedusers` - FTP proxy configuration file to define user access

To use these samples, copy them into the `/etc` directory.

In `/usr/lpp/fw/etc/security` you can find the firewall default definitions:

- ▶ `fwaudio.cfg` - Real Audio definition
- ▶ `fwdaemon.cfg` - daemons default values
- ▶ `fwobjects.cfg` - network objects definition
- ▶ `fwrules.cfg` - default rules definition
- ▶ `fwservices.cfg` - default services definitions
- ▶ `fwsocks.cfg` - default SOCKS definitions
- ▶ `logmgmt.cfg` - log facilities

To use these samples, copy them into the `/etc/security` directory.

To use the GUI interface, you have to copy the `/usr/lpp/fw/etc/security/fwguicmds.En_US` (`.Ja_JP` for Japanese version) in the `/etc/security` directory.

8.7 RACF considerations

z/OS Firewall Technologies has several external security manager considerations, so please read carefully the External Security Manager section in *OS/390 SecureWay Security Server Firewall Technologies Guide and Reference*, SC24-5835 to be sure all definitions are present in your system.

8.7.1 Configuring TCP/IP on the firewall host

To use the firewall services on your TCP/IP, you have to do the following administration tasks on each stack you want to control:

► In the TCP/IP profile:

– Add PORT reserves as follows:

```
PORT
  514  UDP 0MVS           ; firewall syslog
  1014 TCP 0MVS           ; firewall configuration server
  1080 TCP 0MVS           ; firewall socket server
```

– Add the following to the IPCONFIG statement:

```
IPCONFIG FIREWALL DATAGRAMFWD
```

After that you have to recycle (stop/start) TCP/IP.

For more information about configuration statements please refer to *z/OS V1R2.0 CS: IP Configuration Reference*, SC31-8776.

To start firewall servers, we use the procedure shown in Figure 8-1 on page 104 (a sample is provided in the installation data set).

```

//*****
/* PURPOSE: This is a customized version of the Firewall kernel      *
/*          proc used to run the Firewall code in IBM                  *
//*****
/*
//FWKERN PROC  REGSIZE=0M,
//             OUTCLASS='A',
//             LP='',
//             PARS=' '
/*
//*****
/* Start the Firewall kernel                                          *
//*****
/*
//GO          EXEC PGM=ICADCT,
//            REGION=&REGSIZE,
//            TIME=1440,
//            PARM=('&LP/&PARMS >DD:ctout')
/*
//STEPLIB DD DISP=SHR,DSN=ICA.SICALMOD
//CEEDUMP DD SYSOUT=&OUTCLASS
//SYSDUMP DD SYSOUT=&OUTCLASS
//SYSOUT  DD SYSOUT=&OUTCLASS
//CTOUT   DD SYSOUT=&OUTCLASS,DCB=LRECL=250
//CTDUMP  DD SYSOUT=&OUTCLASS
//CTSTG   DD SYSOUT=&OUTCLASS
//LOGDOUT DD SYSOUT=&OUTCLASS,DCB=LRECL=250
//LOGDDUMP DD SYSOUT=&OUTCLASS
//LOGDSTG DD SYSOUT=&OUTCLASS
//SOCDOUT DD SYSOUT=&OUTCLASS,DCB=LRECL=250
//SOCDDUMP DD SYSOUT=&OUTCLASS
//SOCDSTG DD SYSOUT=&OUTCLASS
//FTPDOUT DD SYSOUT=&OUTCLASS,DCB=LRECL=250
//FTPDDUMP DD SYSOUT=&OUTCLASS
//FTPSTG  DD SYSOUT=&OUTCLASS
//FLOGOUT DD SYSOUT=&OUTCLASS,DCB=LRECL=250
//FLOGDUMP DD SYSOUT=&OUTCLASS
//FLOGSTG DD SYSOUT=&OUTCLASS
//TNATOUT DD SYSOUT=&OUTCLASS,DCB=LRECL=250
//TNATDUMP DD SYSOUT=&OUTCLASS
//TNATSTG DD SYSOUT=&OUTCLASS
/*
//*****
/* TCP/IP Data set                                                  *
//*****
/*
//SYSTCPD DD DSN=TCP.TCPPARMS(TDATA28C),DISP=SHR

```

Figure 8-1 Firewall procedure

8.8 Configuring and using the configuration server and client (GUI)

z/OS Firewall Technologies provides a user-friendly interface to manage the entire firewall environment. In fact we can administer this product through a Java-based graphical user interface called the configuration client. This client allows an administrator to perform remote configuration and administration. The client runs on AIX, Windows 95, or Windows NT.

To ensure privacy and integrity, the configuration connection is authenticated using RACF and all data is protected by the Secure Sockets Layer (SSL).

For any prerequisite information and software level, please read Chapter 2 of *OS/390 SecureWay Security Server Firewall Technologies Guide and Reference*, SC24-5835.

8.8.1 Simple configuration scenario

This section describes how we used the GUI to define a virtual private network (VPN) between a z/OS and an RS/6000, both with the firewall product installed.

Figure 8-2 shows our test environment.

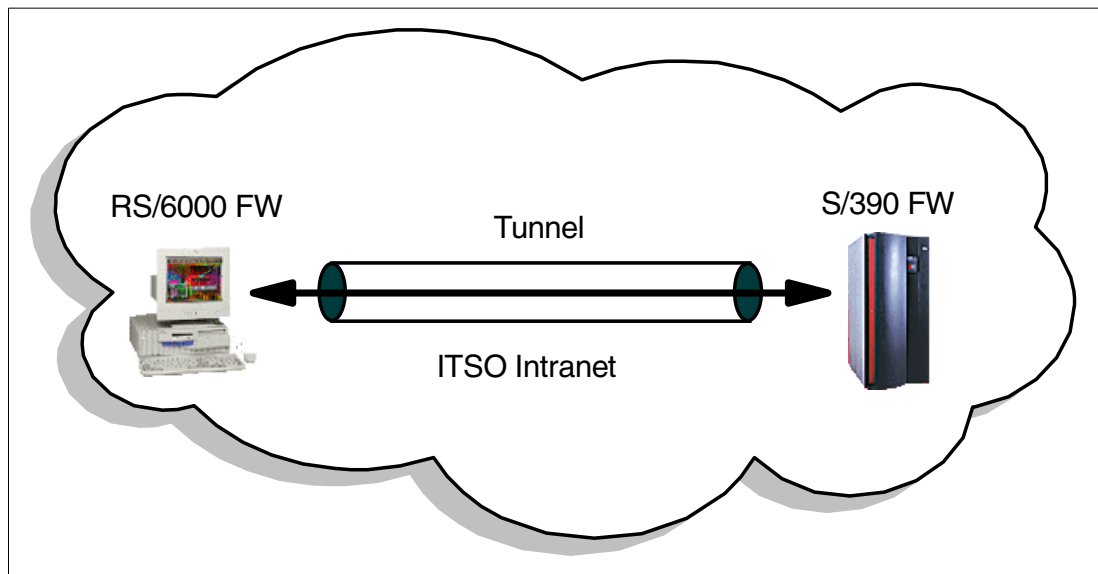


Figure 8-2 Simple test scenario

8.8.2 Configuring SSL

Before starting to use the GUI, the administrator has to configure the SSL environment. From OMVS, the `gskkyman` command must be run to create a new key database, create and store a Version 3 self-signed certificate, and then store the encrypted database password. If you are using z/OS V1R2 or later, you can also store digital certificates in the RACF database for the firewall GUI. Please refer to Chapter 9, “Tools for application security” on page 185 for more information.

For detailed information on how to run this command and the prerequisites, see the *OS/390 System Secure Sockets Layer Programming Guide and Reference*, SC24-5877.

Figure 8-3 shows the main panel for the `gskkyman` command used to set up digital certificates on z/OS.

```
CAMILUC @ RA28:/usr/lpp/fw/bin>gskkyman

          IBM Key Management Utility

Choose one of the following options to proceed.

  1 - Create new key database
  2 - Open key database
  3 - Change database password

  0 - Exit program

Enter your option number:
```

Figure 8-3 Key management utility

8.8.3 Configuring the configuration server (CFGSRV)

The configuration server must be configured to use the SSL option; this is the only way to allow the client to connect to the server.

To configure the CFGSRV we used the `fwdaemon` command (another way is to edit manually the `fwdaemon.cfg` file). The necessary steps are:

1. Specify the key database name (the one you created in the previous section) and assign a port number (the same as specified in the TCP/IP profile):

```
fwdaemon cmd=change daemon=CFGSRV daemonopts="-f /dir/key.kdb -p 1014'
```

2. Configure the server for starting by FWKERN:

```
fwdaemon cmd=change started=yes daemon=CFGSRV
```

FWKERN will now start the configuration server at startup. If FWKERN is already running, you can start the server manually using:

- ▶ From OMVS:

```
fwdaemon cmd=start daemon=CFGSRV
```

- ▶ From the z/OS console:

```
F FWKERN,START CFGSRV
```

For more details about these commands, please refer to *OS/390 SecureWay Security Server Firewall Technologies Guide and Reference*, SC24-5835.

8.8.4 Setting up the configuration client on Windows

The client code is available for the AIX and Windows environment. In our implementation, we used a client installed in a Windows NT 4.0 machine.

These are the necessary steps:

1. Download the code file `/usr/lpp/fw/bin/fwtech.zip` on your workstation.
2. Unzip the code file in a directory of your choice.

3. Change the directory to select the appropriate locale (in our case en_US).
4. Access the client directory and run the Setup command.
5. Follow the instructions given in the window.

8.8.5 Accessing the configuration client (GUI)

To use the GUI you must log in from the workstation where you installed the client code. Your user must be defined according to the RACF definitions reported in *OS/390 SecureWay Security Server Firewall Technologies Guide and Reference*, SC24-5835.

Launch the client GUI (from the Windows start menu). The following window appears.

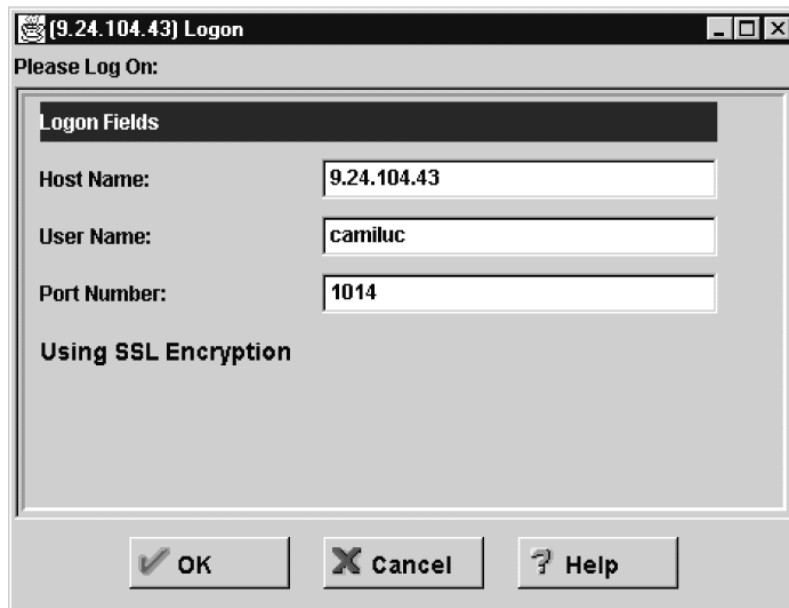


Figure 8-4 Firewall GUI logon window

1. Enter the IP address or the name of the z/OS configuration server.
2. Enter your user ID.
3. Specify the port number you defined.
4. You are prompted for a password.
5. Click **OK**.

If the data you entered is valid, you see the main configuration window.



Figure 8-5 Firewall GUI main window

The configuration client has an easy tree-style navigation. To access the various configuration window, double-click the appropriate folder in the navigation pane on the left side of the window.

General features of the main window

From the main window you can access some useful functions simply by clicking the relevant button:

- ▶ Logoff/Logon
You can log off/log on to any remote server using the appropriate user ID.
- ▶ Help
An HTML help window is displayed, which contains a lot of useful information about your firewall configuration.
- ▶ Log Viewer
The log viewer allows you to browse the firewall logs.
- ▶ Command
You can select **Query Daemon Status**, **Query Daemon Configuration**, and **Query Firewall Stack(s) Status**. The result is shown in the Command Viewer Results area.

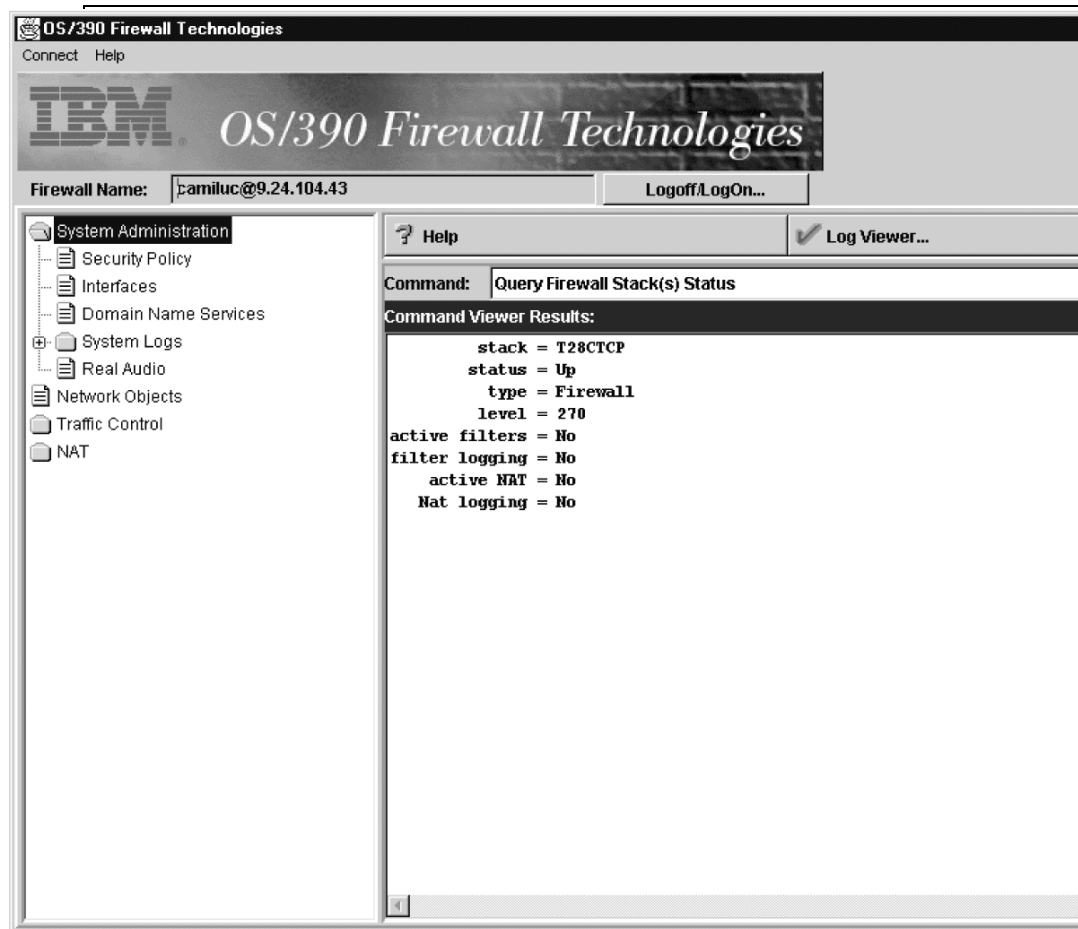


Figure 8-6 Firewall command window

8.8.6 Tunnel definition

In our test scenario we show how to define a tunnel using the new GUI interface.

For testing purposes we defined a tunnel where all kinds of traffic is permitted, but obviously, in a real scenario the security policy requires an in-depth investigation to eliminate any security exposure.

The following is a short description, illustrated with the GUI windows, of the necessary steps to define the tunnel correctly. For additional information about the parameters chosen or how to use the windows, please refer to *OS/390 SecureWay Security Server Firewall Technologies Guide and Reference*, SC24-5835 or the online help.

1. First we add an object to represent the local z/OS (Figure 8-7 on page 110).

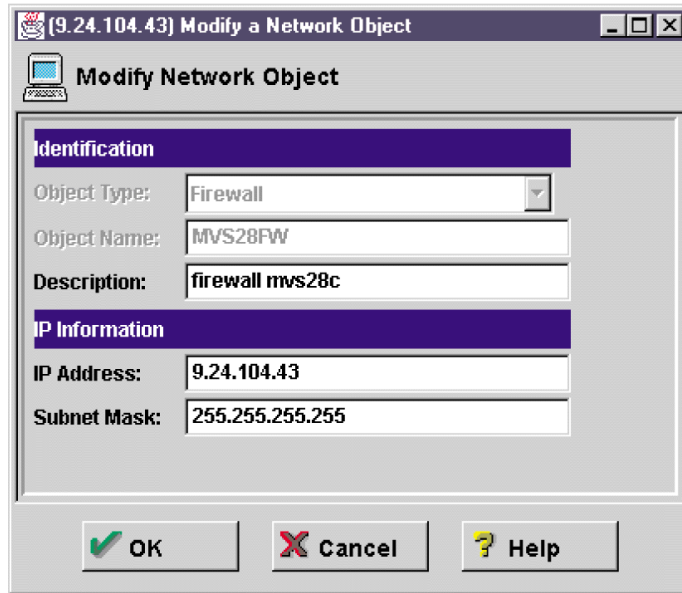


Figure 8-7 z/OS firewall definition

- Next we add an object to represent the remote RS/6000 (Figure 8-8).

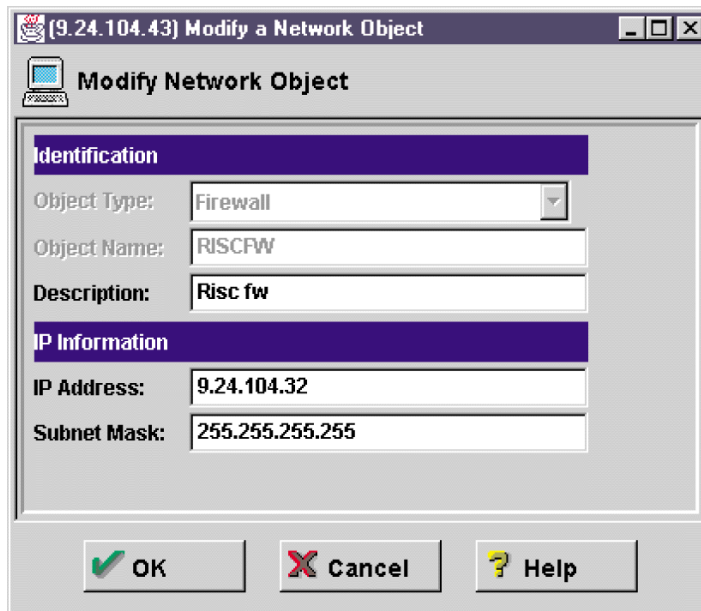


Figure 8-8 RS6000 firewall definition

- As shown in Figure 8-9 on page 111, we define a tunnel between the previously defined objects. For this tunnel we choose both the encryption and authentication policies. The tunnel ID (100) is arbitrarily chosen.

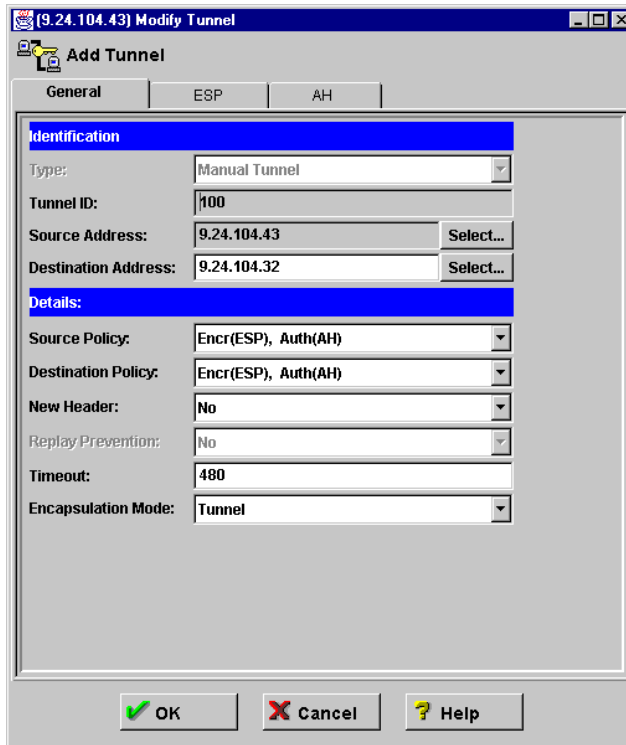


Figure 8-9 General tunnel definition

- Next we define Encapsulation Security Payload (ESP) tunnel parameters: the CDMF algorithm and two arbitrarily chosen Security Parameter Indices (SPI), which must be unique in the environment. We let the system automatically generate the keys. See (Figure 8-10 on page 112).

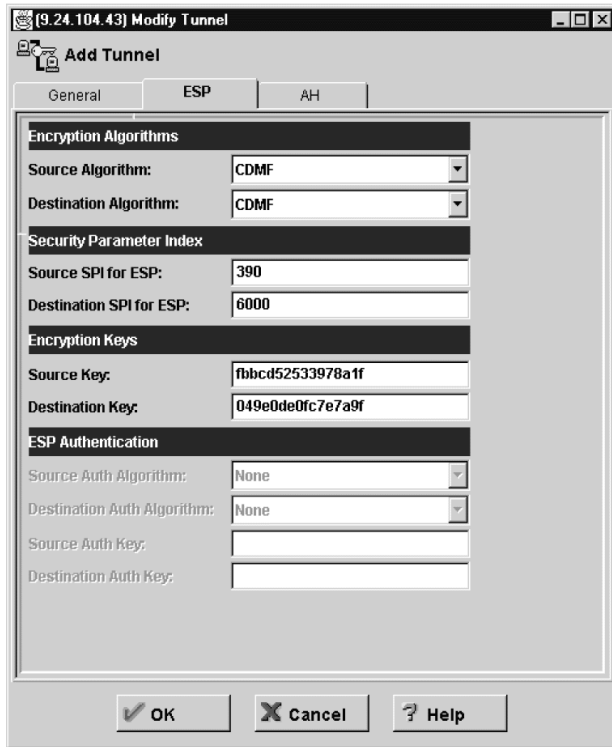


Figure 8-10 ESP tunnel definition

- As shown in Figure 8-11, we define Authentication Header (AH) tunnel parameters: the KEYED_MD5 algorithm and two arbitrarily chosen SPI (they must be unique in the environment). We let the system automatically generate the keys.

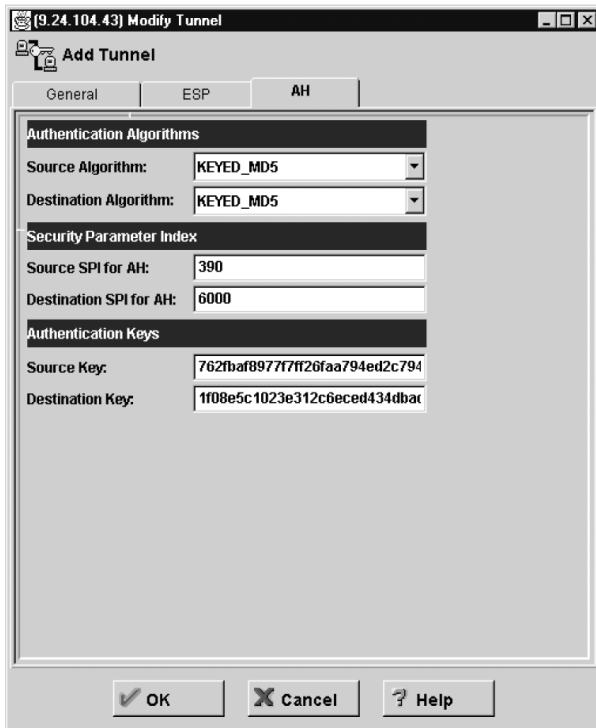


Figure 8-11 AH tunnel definition

6. In Figure 8-12, we define a rule that allows all kinds of traffic inside the tunnel we created before.

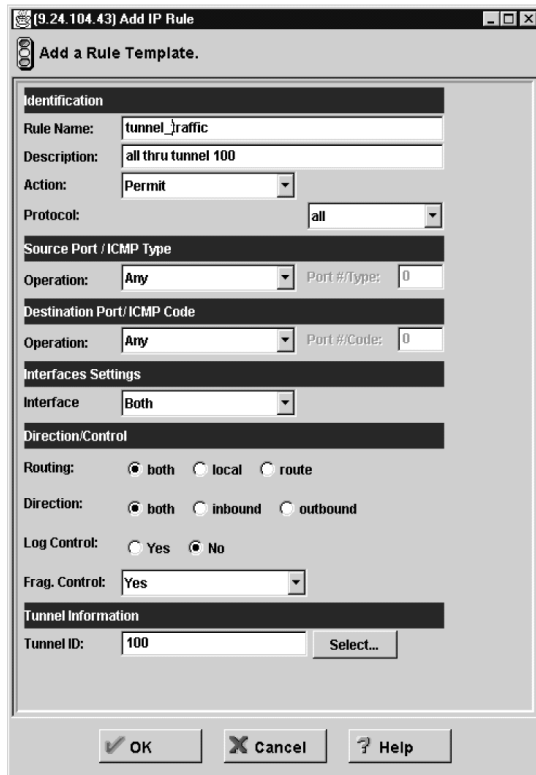


Figure 8-12 Rule for general traffic

7. In Figure 8-13 on page 114, we define a rule that allows the Encapsulating Security Payload (ESP) protocol to flow between the two systems.

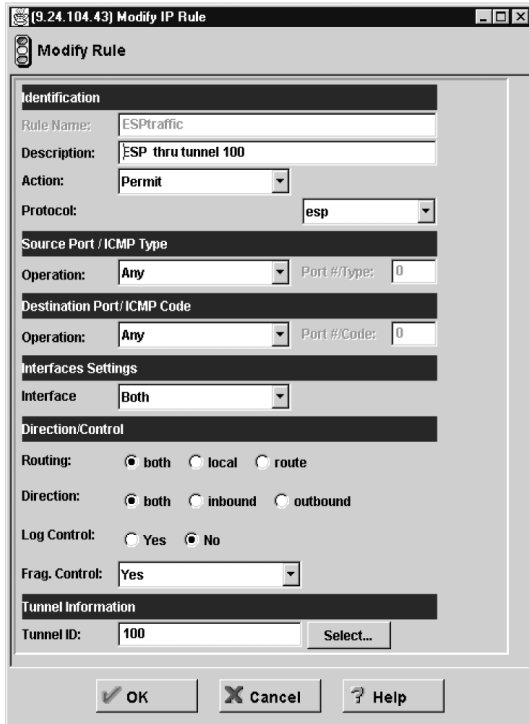


Figure 8-13 Rule for ESP traffic

8. As shown in Figure 8-14, we define a rule that allows the Authentication Header (AH) protocol to flow between the two systems.

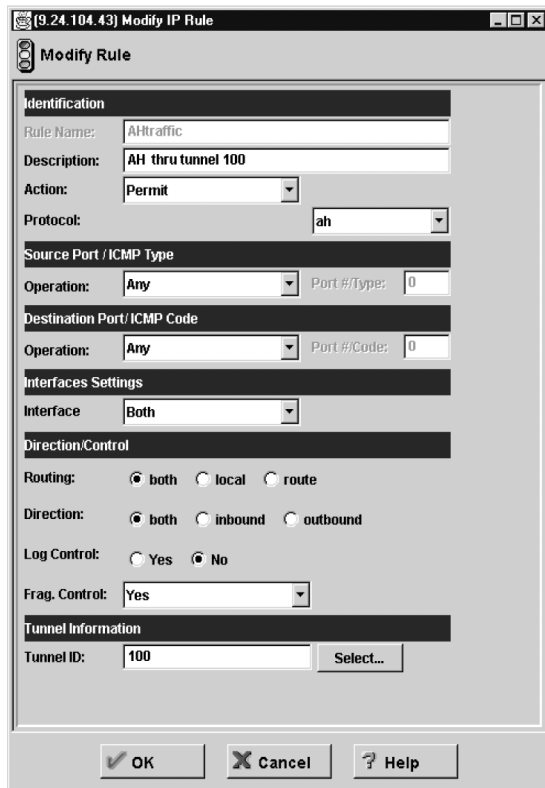


Figure 8-14 Rule for AH traffic

- In Figure 8-15, we define a service to combine the filters rule we defined before. It describes the permitted inbound and outbound traffic in the tunnel.

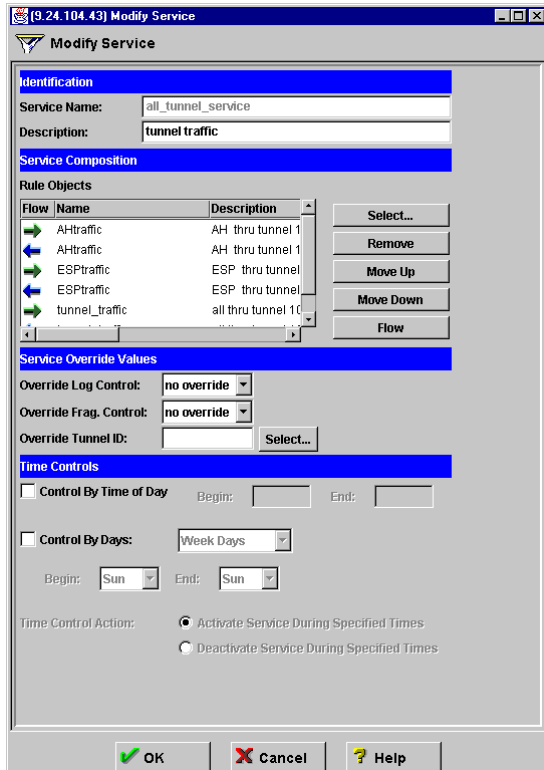


Figure 8-15 Firewall service definition

- In Figure 8-16 on page 116, we define a connection to associate the local and remote objects with the filter rule service.

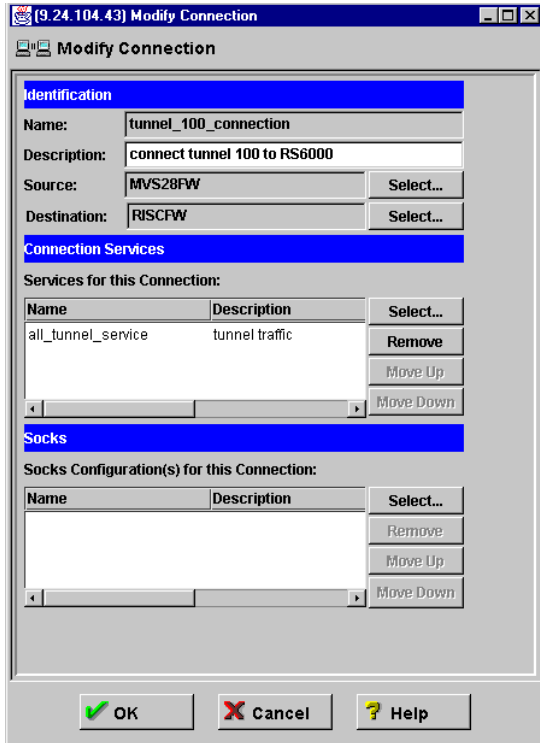


Figure 8-16 Firewall connection definition

11. Next we activate the tunnel definition (Figure 8-17).

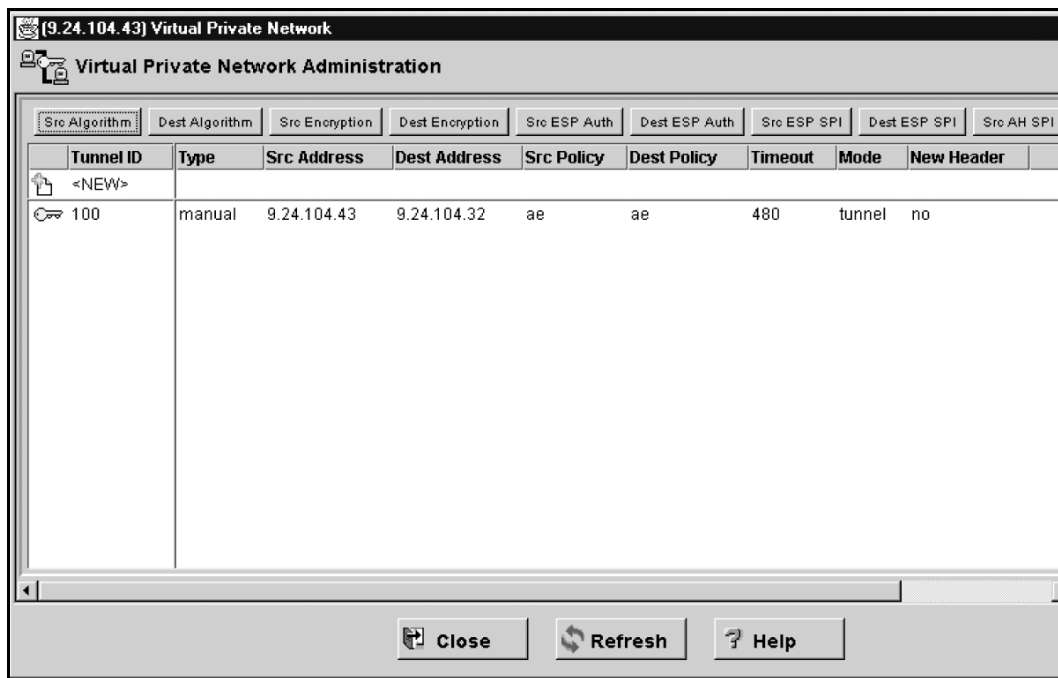


Figure 8-17 Tunnel activation

12. In Figure 8-19 on page 117, we validate the filter rule we created.

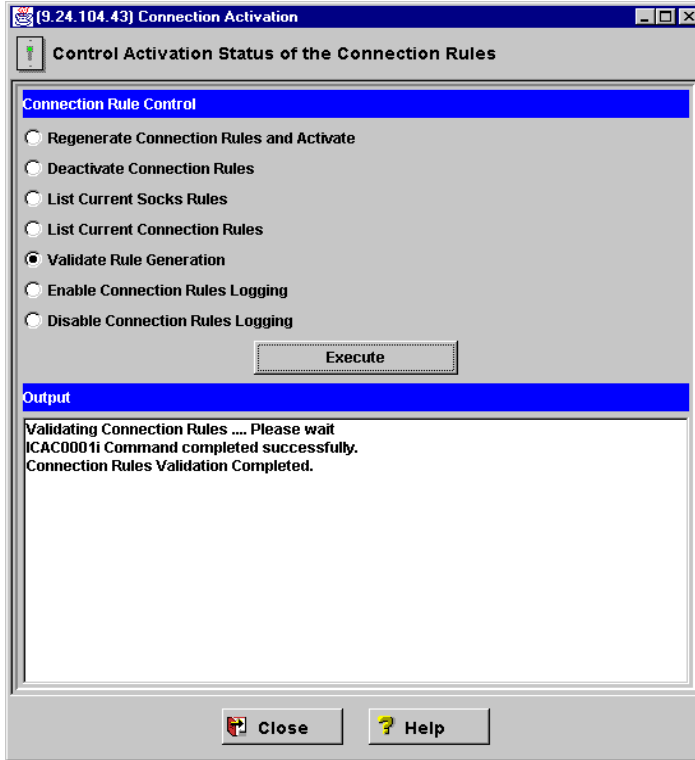


Figure 8-18 Validating rules

13.Last, in Figure 8-19, we activate the filter rule.

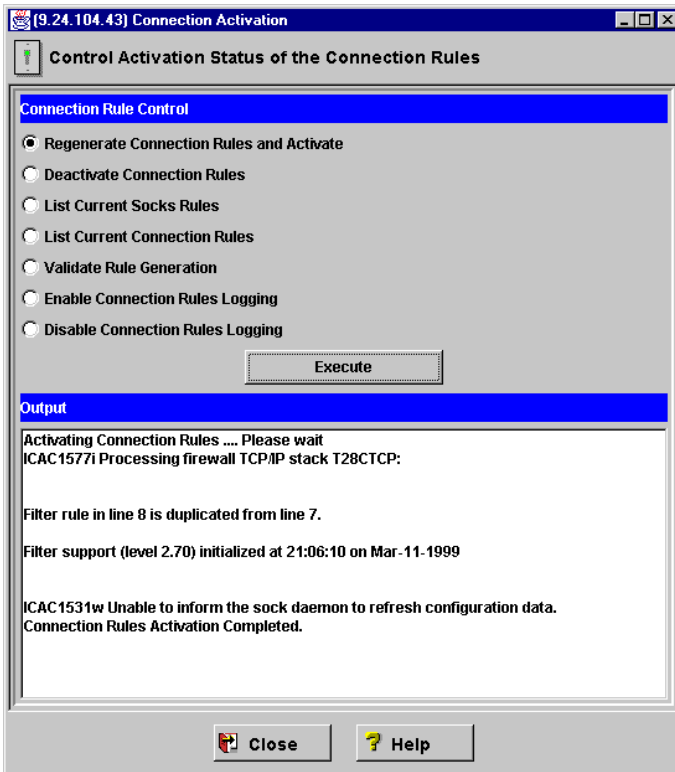


Figure 8-19 Firewall rule activation

The same step should be repeated, of course, on the RS/6000 to reflect the parameters we chose for the local system.

If you have made some mistakes in the definitions (for example, you couldn't access the GUI window to make adjustments) you would have to issue commands from the OMVS shell. To deactivate the wrong filter rule you must use the following command:

```
fwfilter cmd=shutdown
```

After the successful completion of the command, you can reconnect the GUI.

8.8.7 FWTUNNL export file conversion from z/OS and AIX

To adapt the OS/390 tunnel export file to AIX 4.3.2 format and to allow us to import it to the RS/6000 side, it was necessary to modify some fields. We didn't find any guide for this, so the conversion was done matching manually the export files obtained in the two different environments.

Figure 8-20 and Figure 8-21 on page 119 are the export files before and after the conversion to the AIX format.

```
#Tunnel_partner Version = 1.0
9.24.104.43
9.24.104.32
100
6000
6000
390
390
CDMF
8
0x049e0de0fc7e7a9f
CDMF
8
0xfbbcd52533978a1f
KEYED_MD5
16
0x1f08e5c1023e312c6eced434dbad9adb
KEYED_MD5
16
0x762fbaf8977f7ff26faa794ed2c79475
0
28800
ESP_NOOP
0
0x
ESP_NOOP
0
0x
n
n
n
n
n
n
n
n
```

Figure 8-20 Export file in z/OS format

```

#Tunnel_partner Version = 1.0
4
9.24.104.43
9.24.104.32
100
6000
6000
390
390
CDMF
8
0x049e0de0fc7e7a9f
CDMF
8
0xfbbcd52533978a1f
KEYED_MD5
16
0x1f08e5c1023e312c6eced434dbad9adb
KEYED_MD5
16
0x762fba8977f7ff26faa794ed2c79475
0
28800
tunnel
tunnel
eaea
0
0
NONE
0

NONE
0

0
-
-

```

Figure 8-21 Export file in AIX format

8.8.8 On-demand dynamic tunnels scenario

For the dynamic tunnels, we created two tunnels applying an on-demand tunnel feature. Once you define the connection and a packet that matches that particular anchor rule arrives, the tunnel is created automatically by the firewall and all the following traffic flows throughout the tunnel.

We create a tunnel between two OS/390 images and a tunnel between an OS/390 image and a Check Point Firewall-1 firewall running in a Windows NT workstation. We used the GUI configuration client to configure the OS/390 side of the tunnels in both images. To configure the firewall on Windows NT we used a GUI provided by Firewall-1 called Security Policy Editor.

The scenario is shown in Figure 8-22 on page 120.

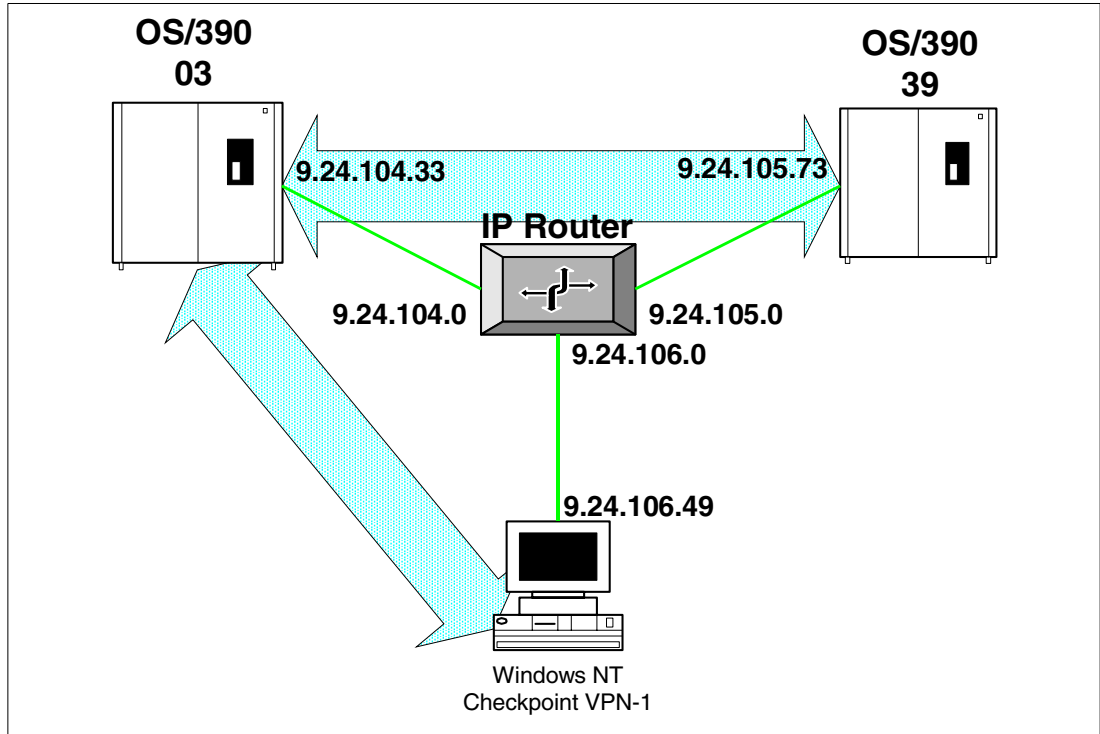


Figure 8-22 Firewall dynamic tunnels scenario

For more information about the firewall configuration and dynamic tunnels, refer to *Secureway Communications Server for OS/390 V2R8 TCP/IP: Guide to Enhancements*, SG24-5631.

On-demand tunnels scenario OS/390RA03 configuration

Now you have to define how the tunnel is configured. To configure a tunnel you have a lot of options that must be defined before you start, for example, tunnel mode, encryption algorithms, authentication, and so on. To make things easier we have created a table that contains all the definitions that are necessary to configure a tunnel between two machines from an OS/390 firewall perspective.

Table 8-1 on page 121, Table 8-2 on page 122, and Table 8-2 on page 122 contain the definitions for the tunnels that are defined.

Table 8-1 VPN planning worksheet - zSeries and NT Firewall-1

VPN Parameter	Value
Key Policy, Proposal, Transform:	
Initiator Negotiation	Main
Responder Negotiation	Main
Authentication Method	Pre-Shared Keys
Hash Algorithm	SHA
Encryption Algorithm	3DES_CBC_8
Diffie-Hellman Group	Group 1
Maximum Key Lifetime	1440
Maximum Size Limit	1000
Key Lifetime Range	60-1440
Size Limit Range	1-1000
Data Policy, Proposal, AH and ESP Transform:	
PFS (Perfect Forward Secrecy)	Group 1
AH Encapsulation Mode	Not applicable
AH Authentication Algorithm	Not applicable
AH Maximum Data Lifetime	Not applicable
AH Maximum Size Limit	Not applicable
AH Data Lifetime Range	Not applicable
AH Size Limit Range	Not applicable
ESP Encapsulation Mode	Tunnel
ESP Authentication Algorithm	HMAC_SHA
ESP Encryption Algorithm	3DES_CBC_8
ESP Maximum Data Lifetime	60
ESP Maximum Size Limit	50000
ESP Data Lifetime Range	60-480
ESP Size Limit Range	1-50000
Dynamic Tunnel Policy:	
Initiation	Either
Connection Lifetime	0
Authentication Information:	
Remote Key Server	9.24.106.49
Authentication Method	Pre-Shared Keys
Shared Key	616263646566 (abcdef in ascii)

VPN Parameter	Value
Certificate Authority:	
RACDCERT Label	Not applicable
Key Ring:	
User ID	Not applicable
Key Ring Name	Not applicable
On-demand Dynamic Connection:	
Source IP Granularity	Packet
Destination IP Granularity	Packet
Gateway Key Server	9.24.106.49
Key Servers:	
Local Key Server ID Type	IPV4
Local Key Server ID	9.24.104.33
Remote Key Server ID Type	IPV4
Remote Key Server ID	9.24.106.49

Table 8-2 VPN planning worksheet - z/OS RA03 and z/OS 39

VPN Parameter	Value
Key Policy, Proposal, Transform:	
Initiator Negotiation	Main
Responder Negotiation	Main
Authentication Method	RSASIG
Hash Algorithm	SHA
Encryption Algorithm	3DES_CBC_8
Diffie-Hellman Group	Group 1
Maximum Key Lifetime	1440
Maximum Size Limit	1000
Key Lifetime Range	60-1440
Size Limit Range	1-1000
Data Policy, Proposal, AH and ESP Transform:	
PFS (Perfect Forward Secrecy)	Group 1
AH Encapsulation Mode	Transport
AH Authentication Algorithm	SHA
AH Maximum Data Lifetime	60
AH Maximum Size Limit	0

VPN Parameter	Value
AH Data Lifetime Range	60-480
AH Size Limit Range	0-0
ESP Encapsulation Mode	Transport
ESP Authentication Algorithm	HMAC_SHA
ESP Encryption Algorithm	3DES_CBC_8
ESP Maximum Data Lifetime	60
ESP Maximum Size Limit	50000
ESP Data Lifetime Range	60-480
ESP Size Limit Range	1-50000
Dynamic Tunnel Policy:	
Initiation	Either
Connection Lifetime	0
Authentication Information:	
Remote Key Server	9.24.105.73
Authentication Method	RSASIG
Shared Key	Not applicable
Certificate Authority:	
RACDCERT Label	MVS039B CA
Key Ring:	
User ID	fwkern
Key Ring Name	R2615.IkeKeyRing03B
On-demand Dynamic Connection:	
Source IP Granularity	Anchor
Destination IP Granularity	Anchor
Gateway Key Server	Not applicable
Key Servers:	
Local Key Server ID Type	IPV4
Local Key Server ID	9.24.104.33
Remote Key Server ID Type	IPV4
Remote Key Server ID	9.24.105.73

Table 8-3 VPN planning worksheet - z/OS 39 and z/OS RA03

VPN Parameter	Value
Key Policy, Proposal, Transform:	
Initiator Negotiation	Main
Responder Negotiation	Main
Authentication Method	RSASIG
Hash Algorithm	SHA
Encryption Algorithm	3DES_CBC_8
Diffie-Hellman Group	Group 1
Maximum Key Lifetime	1440
Maximum Size Limit	1000
Key Lifetime Range	60-1440
Size Limit Range	1-1000
Data Policy, Proposal, AH and ESP Transform:	
PFS (Perfect Forward Secrecy)	Group 1
AH Encapsulation Mode	Transport
AH Authentication Algorithm	SHA
AH Maximum Data Lifetime	60
AH Maximum Size Limit	0
AH Data Lifetime Range	60-480
AH Size Limit Range	0-0
ESP Encapsulation Mode	Transport
ESP Authentication Algorithm	HMAC_SHA
ESP Encryption Algorithm	3DES_CBC_8
ESP Maximum Data Lifetime	60
ESP Maximum Size Limit	50000
ESP Data Lifetime Range	60-480
ESP Size Limit Range	1-50000
Dynamic Tunnel Policy:	
Initiation	Either
Connection Lifetime	0
Authentication Information:	
Remote Key Server	9.24.104.33
Authentication Method	RSASIG
Shared Key	Not applicable

VPN Parameter	Value
Certificate Authority:	
RACDCERT Label	MVS03B CA
Key Ring:	
User ID	fwkern
Key Ring Name	R2615.IkeKeyRing39B
On-demand Dynamic Connection:	
Source IP Granularity	Anchor
Destination IP Granularity	Anchor
Gateway Key Server	Not applicable
Key Servers:	
Local Key Server ID Type	IPV4
Local Key Server ID	9.24.105.73
Remote Key Server ID Type	IPV4
Remote Key Server ID	9.24.104.33

Now we have to define the keyrings and certificates used in the tunnel between OS/390 RA03 and OS/390 39. We define two keyrings and four certificates that are used in this connection. Both system are sharing the RACF database and the definitions that we made in one system are reflected on the other system. We define two Certificate Authority certificates and two personal certificates, each one signed by one of the certificate authorities. We use the RACDCERT command to create the keyrings and the certificates.

The following figure shows all the commands used to define the keyrings and the certificates.

```

racdcert id(fwkern) addring(R2615.IkeKeyRing39B) 1
racdcert id(fwkern) addring(R2615.IkeKeyRing03B) 2

racdcert certauth gencert subjectsdn(cn('mvs03b.itso.ral.ibm.com')
ou('ITSO') o('IBM') l('Raleigh') sp('North Carolina') c('US'))
withlabel('R2615 - MVS03B CA') altname(ip(9.4.104.33) domain('itso.ral.ibm.com')) 3

racdcert certauth gencert subjectsdn(cn('mvs39b.itso.ral.ibm.com')
ou('ITSO') o('IBM') l('Raleigh') sp('North Carolina') c('US'))
withlabel('R2615 - MVS39B CA') altname(ip(9.4.105.73) domain('itso.ral.ibm.com')) 4

racdcert id(fwkern) gencert subjectsdn(cn('mvs39b.itso.ral.ibm.com') 6
ou('ITSO') o('IBM') l('Raleigh') sp('North Carolina') c('US'))
withlabel('R2615 - MVS39B Isakmpd') altname(ip(9.4.105.73))
signwith(certauth label('R2615 - MVS39B CA'))

racdcert id(fwkern) gencert subjectsdn(cn('mvs03b.itso.ral.ibm.com') 7
ou('ITSO') o('IBM') l('Raleigh') sp('North Carolina') c('US'))
withlabel('R2615 - MVS03B Isakmpd') altname(ip(9.4.105.73))
signwith(certauth label('R2615 - MVS03B CA'))

racdcert id(fwkern) connect(certauth label('R2615 - MVS03B' CA')
ring(R2615.IkeKeyRing03B) usage(certauth))

racdcert id(fwkern) connect(certauth label('R2615 - MVS39B' CA') 8
ring(R2615.IkeKeyRing03B) usage(certauth))

racdcert id(fwkern) connect(certauth label('R2615 - MVS03B' CA') 9
ring(R2615.IkeKeyRing39B) usage(certauth))

racdcert id(fwkern) connect(certauth label('R2615 - MVS39B' CA')
ring(R2615.IkeKeyRing39B) usage(certauth))

racdcert id(fwkern) connect(label('R2615 - MVS03B Isakmpd') default 10
ring(R2615.IkeKeyRing03B) usage(personal))

racdcert id(fwkern) connect(label('R2615 - MVS39B Isakmpd') default 11
ring(R2615.IkeKeyRing39B) usage(personal))

racdcert id(fwkern) listring(*)

```

Ring:

```

>R2615.IkeKeyRing03B<
Certificate Label Name          Cert Owner      USAGE          DEFAULT
-----
R2615 - MVS03B CA              CERTAUTH       CERTAUTH       NO
R2615 - MVS39B CA              CERTAUTH       CERTAUTH       NO
R2615 - MVS03B ISAKMPD         ID(FWKERN)     PERSONAL       YES

```

Ring:

```

>R2615.IkeKeyRing39B<
Certificate Label Name          Cert Owner      USAGE          DEFAULT
-----
R2615 - MVS39B CA              CERTAUTH       CERTAUTH       NO
R2615 - MVS03B CA              CERTAUTH       CERTAUTH       NO
R2615 - MVS39B ISAKMPD         ID(FWKERN)     PERSONAL       YES

```

1 This keyring is used by the OS/390 RA39 TCPIPB. All the keyrings and certificates are associated with the user ID fwkern.

2 This keyring is used by the OS/390 03 TCPIPB.

3 This is the certificate for the MVS03B CA. It is used to sign the ISAKMPD server certificate. The alname ipaddress is required when defining the certificate. It is used later to check the ID of the key server.

4 This is the certificate for the MVS39B CA. It is used to sign the ISAKMPD server certificate. The alname ipaddress is required when defining the certificate. It is used later to check the ID of the key server.

5 This is the certificate used by the ISAKMPD 39 server to identify itself to the other tunnel partners when using RSASIG authentication.

6 This certificate is signed by this CA. The CA certificate must be stored in the partner keyring in order to be able to authenticate this certificate.

7 This is the certificate used by the ISAKMPD RA03 server to identify itself to the other tunnel partners when using RSASIG authentication.

8 We are connecting the CA who signed the 39 partner certificate to the system RA03 keyring. When the RA03 ISAKMPD server receives a certificate to be authenticated, it looks to a specific CA defined in the firewall configuration to authenticate the other partner/peer when using RSASIG authentication.

9 We are connecting the CA who signed the RA03 partner certificate to the system 39 keyring. When the 39 ISAKMPD server receives a certificate to be authenticated, it looks to a specific CA defined in the firewall configuration to authenticate the other partner/peer when using RSASIG authentication.

10 This certificate is being connected to the system RA03 keyring as default. That means that it is used for the system RA03 ISAKMPD server to identify itself when creating a tunnel and using RSASIG authentication.

11 This certificate is being connected to the system 39 keyring as default. That means that it is used for the system 39 ISAKMPD server to identify itself when creating a tunnel and using RSASIG authentication.

The keyring for the system RA03 has a default certificate that is used to identify this peer to the other partners and it contains all the CAs certificates used by the other tunnel partners to authenticate them when using RSASIG authentication.

Now we start the fwkern procedure and check if all the daemons are up and running.

```
S FWKERN
$HASP100 FWKERN  ON STCINRDR
IEF695I START FWKERN  WITH JOBNAME FWKERN  IS ASSIGNED TO USER
FWKERN  , GROUP OMVSGRP
$HASP373 FWKERN  STARTED
IEF403I FWKERN - STARTED - TIME=09.58.42
ICAM1057i Release 2.10.0 Service Level 0000000. Created on Apr 18 319
2000.
$HASP100 ICAPCFGS ON STCINRDR
$HASP373 ICAPCFGS STARTED
IEF403I ICAPCFGS - STARTED - TIME=09.58.43
ICAM1069i Daemon CFGSRV has been started.
$HASP100 ICAPIKED ON STCINRDR
$HASP373 ICAPIKED STARTED
IEF126I GIANCA - LOGGED OFF - TIME=09.59.09
$HASP395 GIANCA  ENDED
IEF403I ICAPIKED - STARTED - TIME=09.58.55
ICAM1069i Daemon ISAKMPD has been started.
$HASP100 ICAPSTAK ON STCINRDR
$HASP373 ICAPSTAK STARTED
IEF403I ICAPSTAK - STARTED - TIME=09.59.10
ICAM1069i Daemon FWSTACKD has been started.
ICAM1003i FWKERN initialization complete.
ICAM1081i IP/SEC initialization complete for TCPIP.
```

As shown, all the daemons are up and running and now you can use the configuration client to configure the tunnels.

We have three systems to be configured. We start by configuring the system RA03.

On-demand tunnels scenario OS/390 RA03 configuration

Start the configuration client on the workstation and log on to the firewall. Expand all the sections you will work with on the window shown in Figure 8-23 on page 129.



Figure 8-23 z/OS firewall main window

This window shows all the traffic control and the virtual private network menus expanded.

Now we show all the windows used to create the objects for this tunnel:

- ▶ Data proposal
- ▶ Data policy
- ▶ Dynamic tunnel policy
- ▶ Key policy
- ▶ Authentication information
- ▶ Key Servers
- ▶ Key server group
- ▶ VPN on-demand
- ▶ Anchor rule
- ▶ Service
- ▶ Connection

The objects we have created are shown in the following figures.

The data proposals

The screenshot shows a window titled "[9.24.104.33] Modify Data Proposal". The main title is "Modify Data Proposal".

Identification

Data Proposal Name: R2615 - Tunnel

Description: Tunnel

AH Transforms

AH Transform Objects:

Name	Description	Select...
		Remove
		Move Up
		Move Down

ESP Transforms

ESP Transform Objects:

Name	Description	Select...
Tunnel/HMAC_SHA/3DES_(Tunnel, HMAC_SHA, 3DES_CB		Remove
		Move Up
		Move Down

Buttons: OK, Cancel, Help

Figure 8-24 z/OS RA03 tunnel to NT data proposal

Note that in this data proposal we selected only the ESP protocol with tunnel mode, 3DES encryption, and SHA authentication as we defined in the table. These options are used in phase 2 to define how the data is protected by the dynamic tunnel.

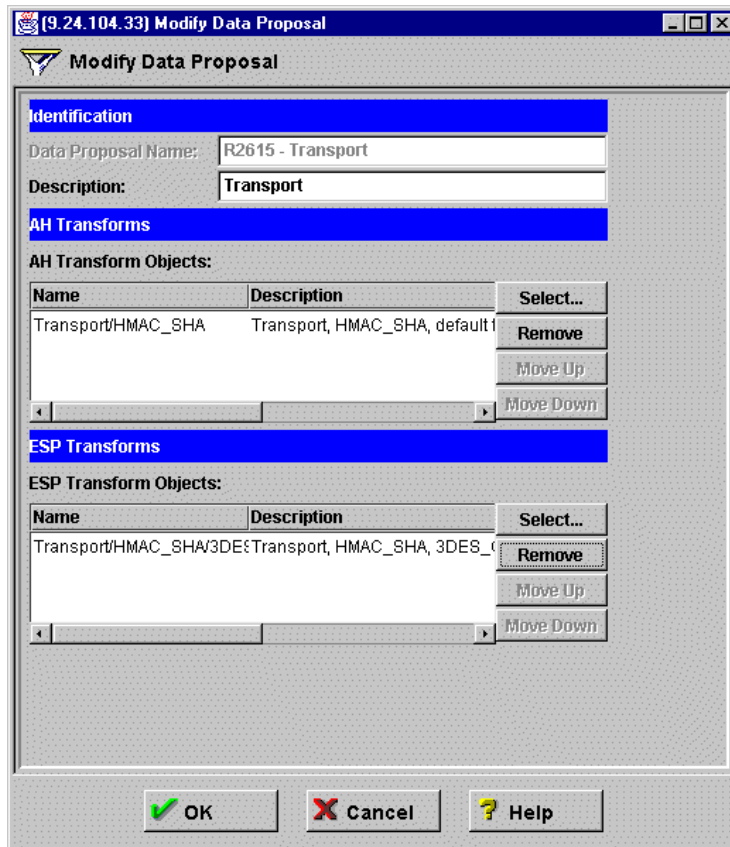


Figure 8-25 zSeries RA03 tunnel to z/OS 39 data proposal

In this data proposal we selected the transport mode for the tunnel and both protocols: AH and ESP. Then we selected 3DES encryption and SHA authentication as we defined in the table.

The data policies

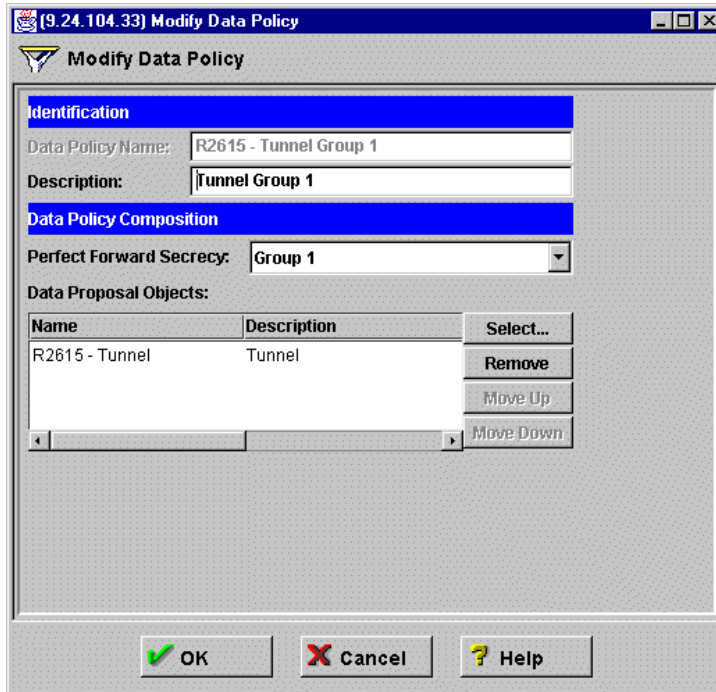


Figure 8-26 z/OS firewall tunnel to NT data policy

For this data policy we selected the tunnel data proposal we created before and we selected Perfect Forward Security Group 1.

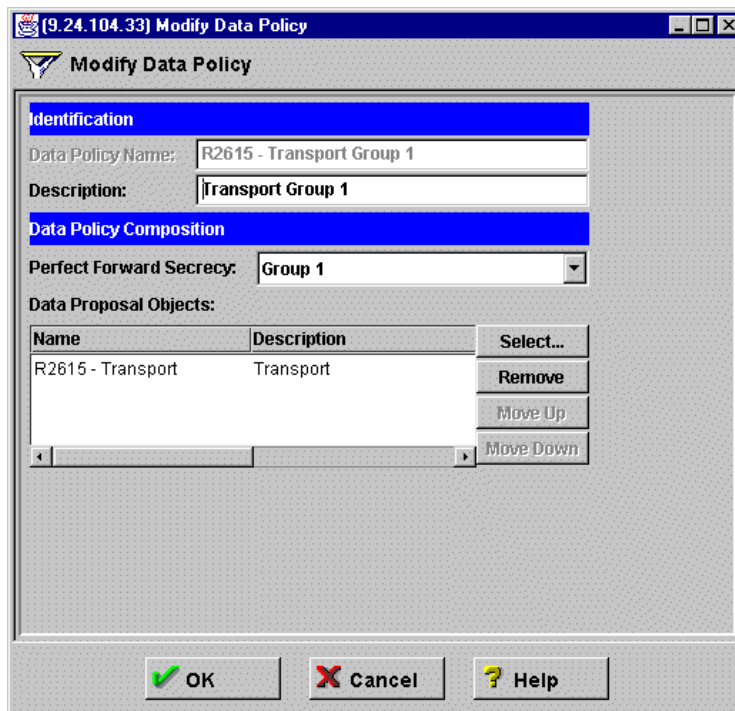


Figure 8-27 zSeries RA03 tunnel to z/OS 39 data policy

For this data policy we selected the transport data proposal we created and selected Perfect Forward Secrecy Group 1.

The dynamic tunnel policy

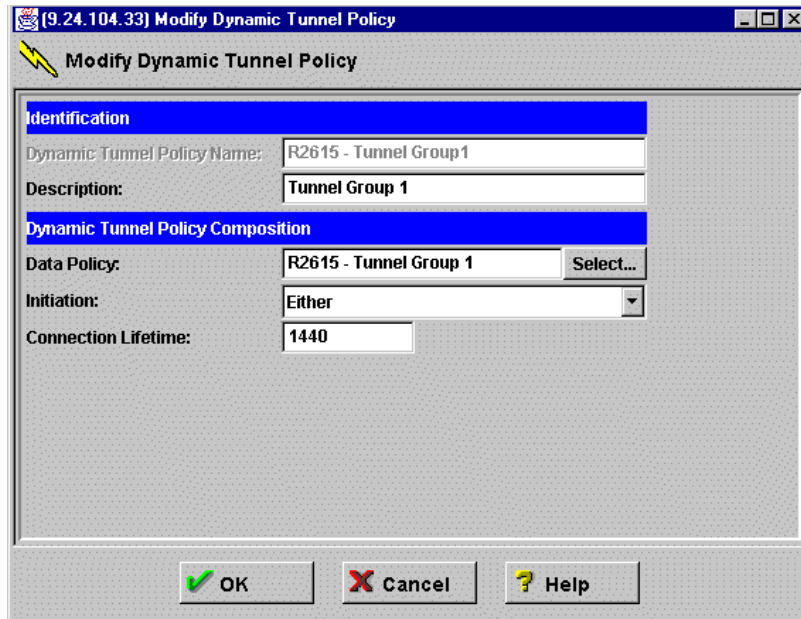


Figure 8-28 z/OS firewall tunnel to NT dynamic tunnel policy

For this dynamic tunnel policy we selected the Tunnel Group 1. The dynamic tunnel policy is associated with the anchor rule.

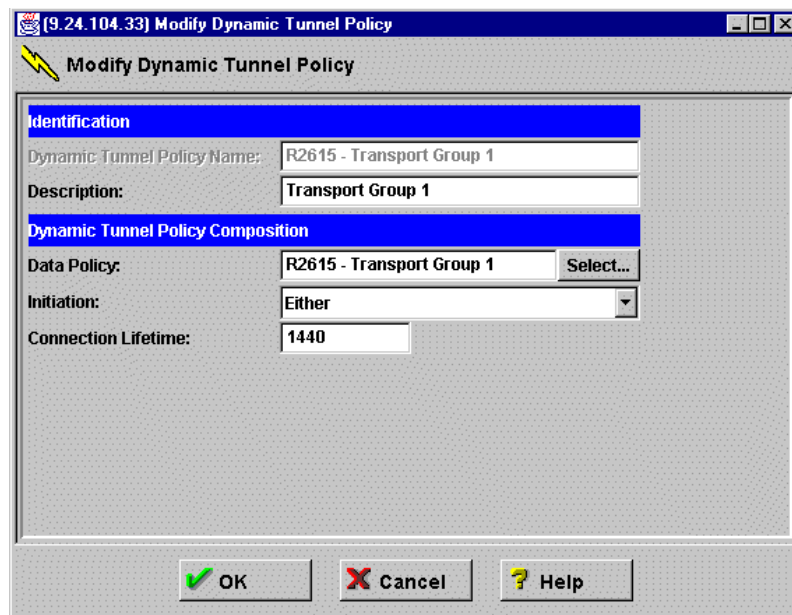


Figure 8-29 zSeries RA03 tunnel to z/OS 39 dynamic tunnel policy

For this dynamic tunnel policy we selected Transport Group 1.

The key policy

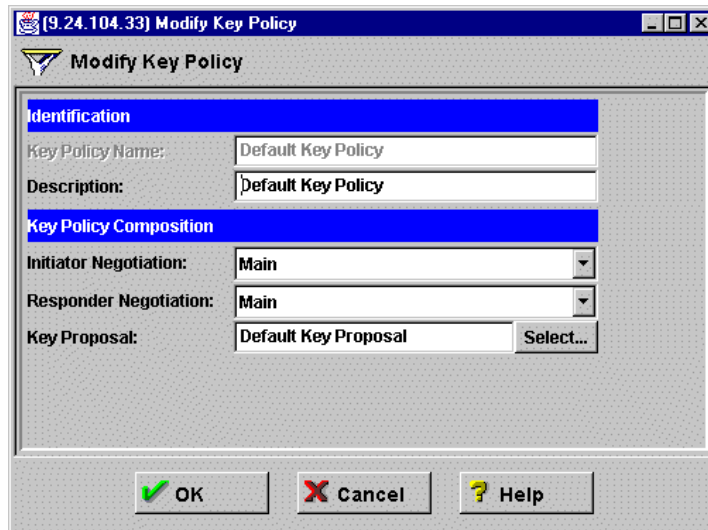


Figure 8-30 z/OS firewall tunnel key policy

For both tunnels we selected the Default Key Policy that comes with z/OS Firewall. Using this default key policy, the z/OS accepts almost any phase 1 negotiation proposal. The first key proposal that matches is used.

Phase 1 creates the security association between the endpoints of the tunnel, and the key policy object defines how the security association is protected.

The key servers

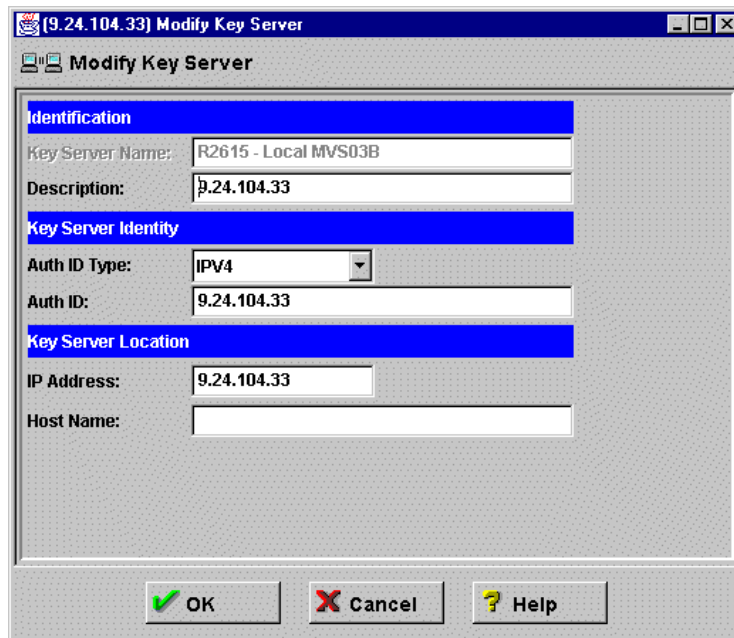


Figure 8-31 z/OS RA03 key server

This is the local key server, the OS/390 RA03 image. The Auth ID 9.24.104.33 is used to check the identity of this server. If you are using digital certificates, the IP address in the digital certificate (altname extension field) is checked against this field to check the authenticity of the partner.

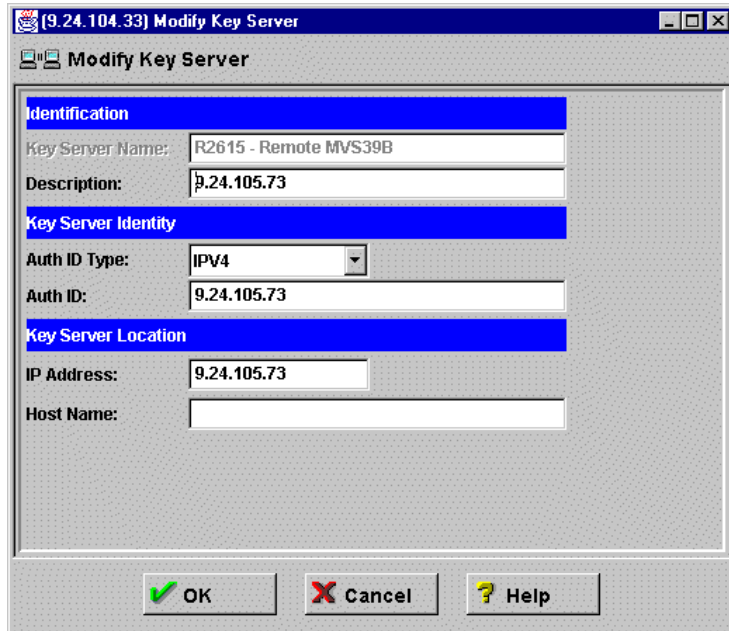


Figure 8-32 z/OS 39 key server

This is the z/OS 39, a remote key server, the other side/peer of the tunnel.

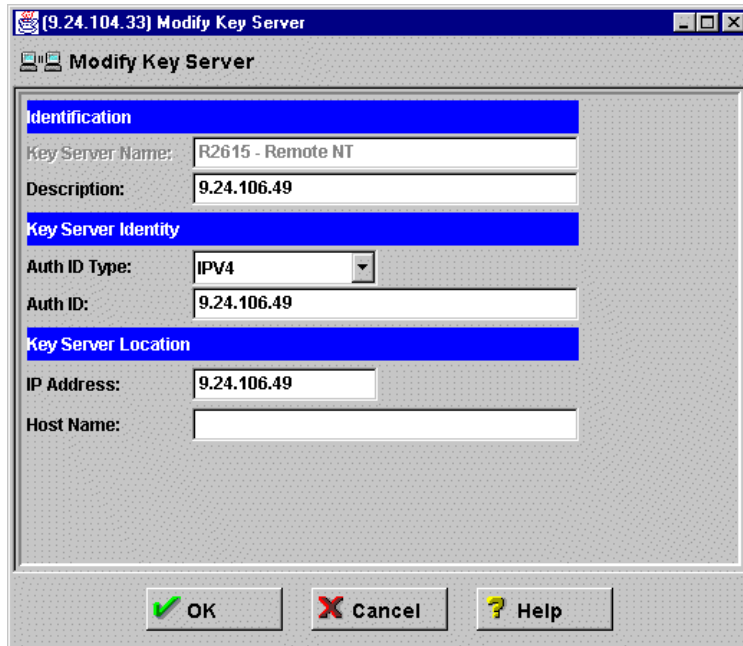


Figure 8-33 NT Firewall-1 key server

This is the Windows NT Firewall-1 machine, a remote key server, the other side/peer of the tunnel.

The key servers group

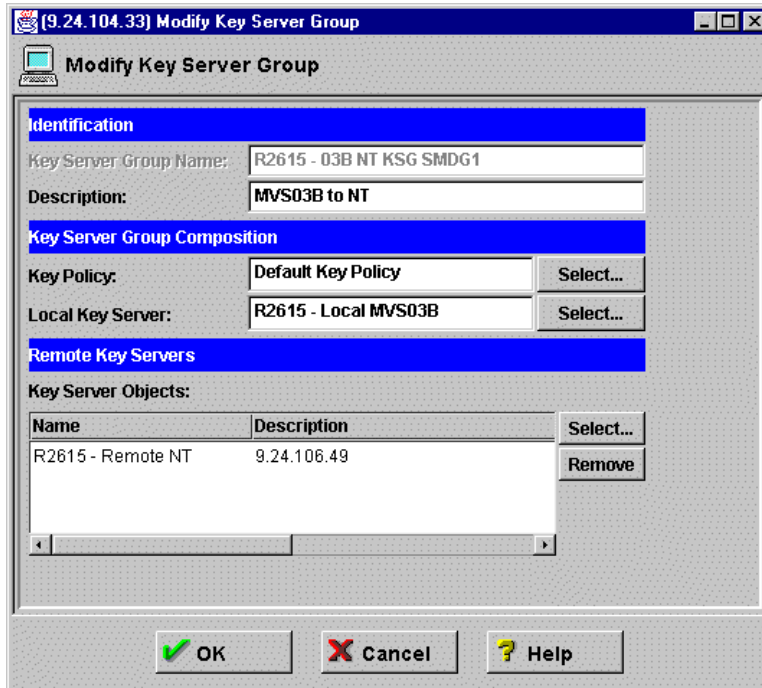


Figure 8-34 z/OS RA03 to NT Firewall-1 key server group

This is the key server group for the tunnel between OS/390 RA03 and the Windows NT Firewall-1 machine. Here we specify the remote key servers and what key policy is used by the local key server with the remote key servers in phase 1.

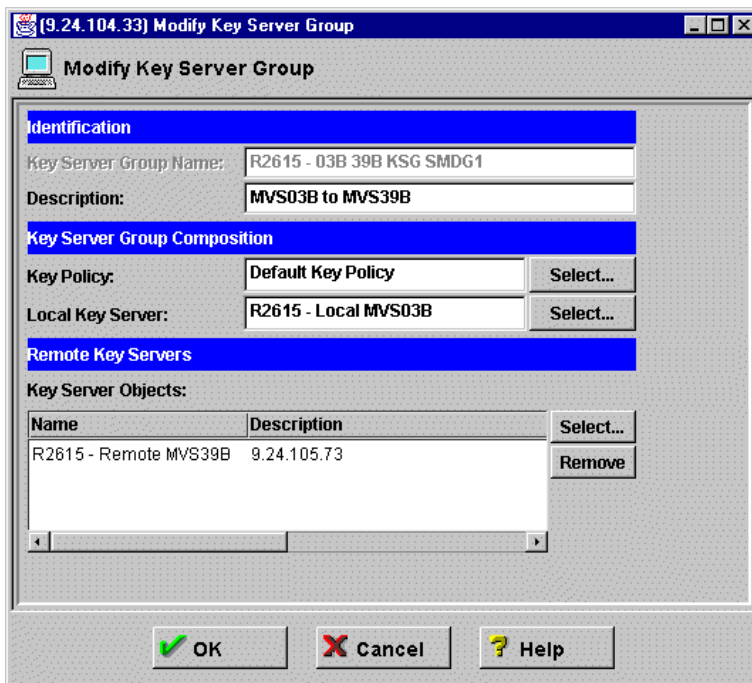


Figure 8-35 z/OS RA03 to z/OS 39 key server group

This is the key server group for the tunnel between OS/390 RA03 and OS/390 39.

The authentication information

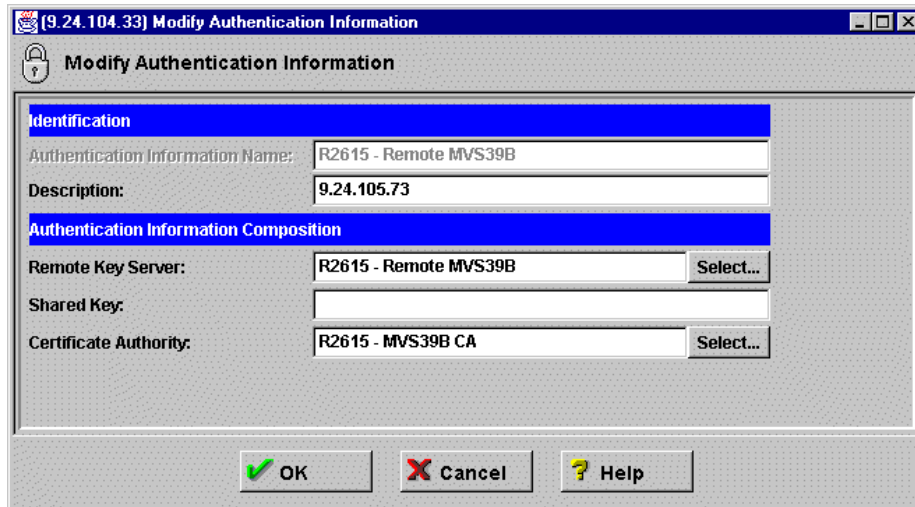


Figure 8-36 z/OS 39 authentication information

This is the authentication information for the OS/390 39. Here we are using a Certificate Authority to authenticate the remote key server. We are saying that we accept any certificate issued by this Certificate Authority. Figure 8-37 shows how to define a keyring and a Certificate Authority.

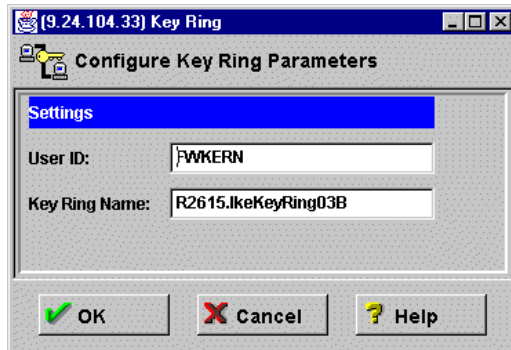


Figure 8-37 Configure Key Ring Parameters window

This keyring must be defined and associated with the user ID listed in the user ID field. The default certificate in this keyring is used to identify this key server in phase 1 negotiation when the RSASIG authentication method is selected. If the other side of the tunnel is an OS/390 firewall, the Certificate Authority who issued/signed this certificate must be connected to this keyring to authenticate the key server. The other certificate authorities or the other peers that are using RSASIG to identify themselves must be connected to this keyring.

Each time you change this information, you have to recycle the ISAKMPD daemon. To do this you can use the `modify fwkern` command to stop and start the daemon at the OS/390 console.

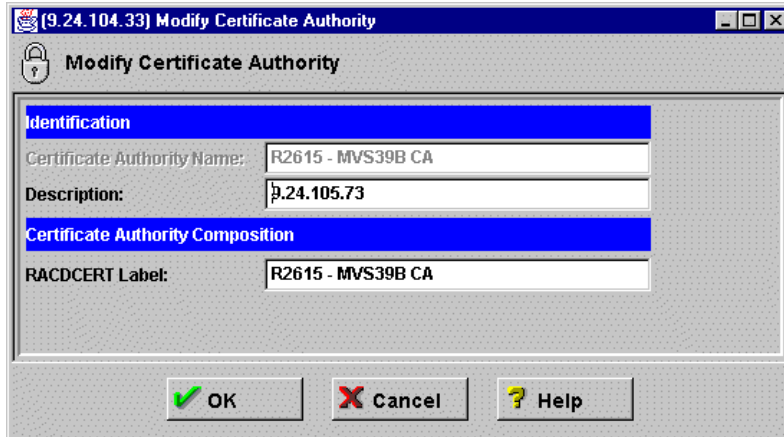


Figure 8-38 Certificate Authority modification

This RACDCERT label represents a certificate in the RACF database and it has to be connected to the keyring defined previously. This Certificate Authority has to be the issuer/signer of the certificate presented by the OS/390 39 firewall on the other side of the tunnel.

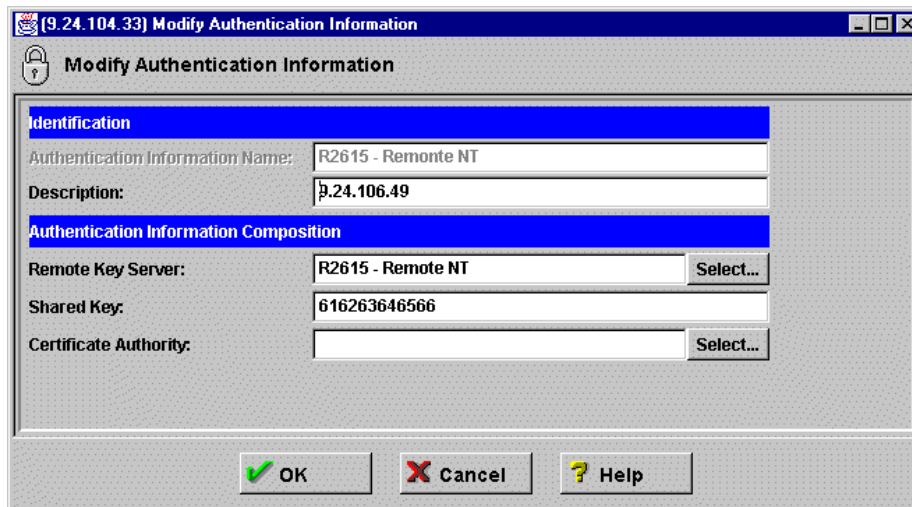


Figure 8-39 NT Firewall-1 authentication information

The tunnel between OS/390 RA03 and Windows NT Firewall-1 is authenticated using pre-shared keys. A pre-shared key is a value that both sides of the tunnel present to each other in order to identify themselves. The value here, 616263646566, is the binary representation of the shared secret key, which is abcdef in ASCII format.

The on-demand object

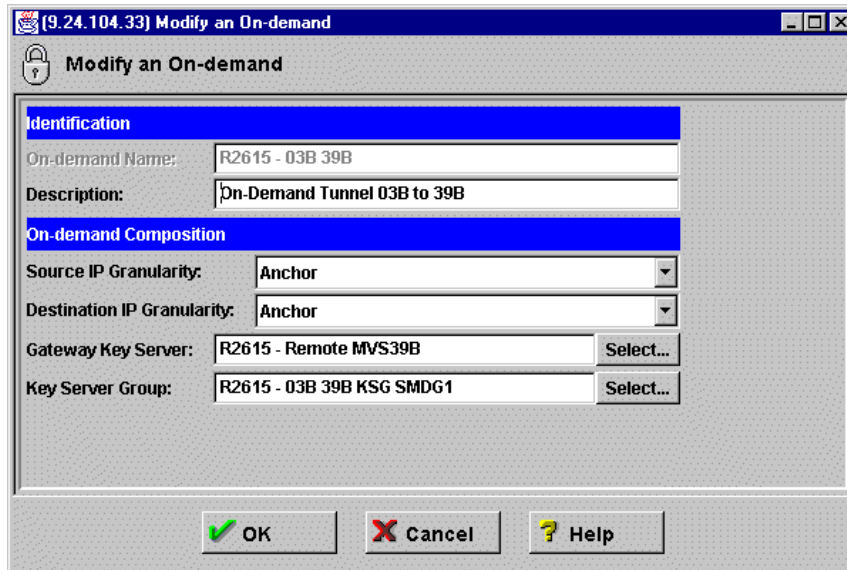


Figure 8-40 z/OS RA03 to z/OS 39 on-demand object

This on-demand object represents the on-demand tunnel definition for the tunnel between OS/390 RA03 and the OS/390 39.

The IP granularity tells the firewall how the dynamic connection is created. It can be based on the source/destination anchor rule or in the source/destination IP packet contents. This value is used as the source/destination value to create the dynamic connection that represents the tunnel.

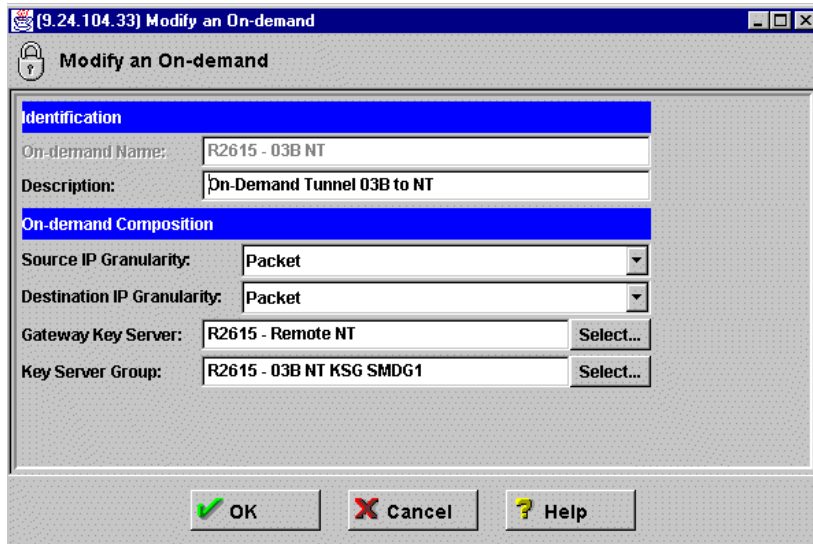


Figure 8-41 z/OS RA03 to NT Firewall-1 on-demand object

This on-demand object represents the on-demand tunnel definition for the tunnel between OS/390 RA03 and the Windows NT Firewall-1 workstation.

The anchor rules

The screenshot shows a 'Modify IP Rule' dialog box with the following configuration:

- Identification:**
 - Rule Name: R2615 - Anchor All Permit EG1T
 - Description: Ether/Group1/Transport
 - Action: Anchor
 - Protocol: all
- Source Port / ICMP Type:**
 - Operation: Any
 - Port #/Type: 0
- Destination Port / ICMP Code:**
 - Operation: Any
 - Port #/Code: 0
- Interfaces Settings:**
 - Interface: Both
- Direction/Control:**
 - Routing: both local route
 - Direction: both inbound outbound
 - Log Control: yes no permit deny
- Tunnel Information:**
 - Manual VPN Tunnel ID: (empty)
 - Dynamic Tunnel Policy Name: R2615 - Transport Group 1

Figure 8-42 z/OS RA03 to z/OS 39 anchor rule

This anchor rule is used to define a tunnel in transport mode allowing all kinds of traffic. It is used between OS/390 RA03 and OS/390 39. Here we are using the dynamic tunnel policy created previously.

When a dynamic VPN connection is activated, the anchor rule is used to determine the placement of the dynamically generated filter rules among the static permit and deny rules.

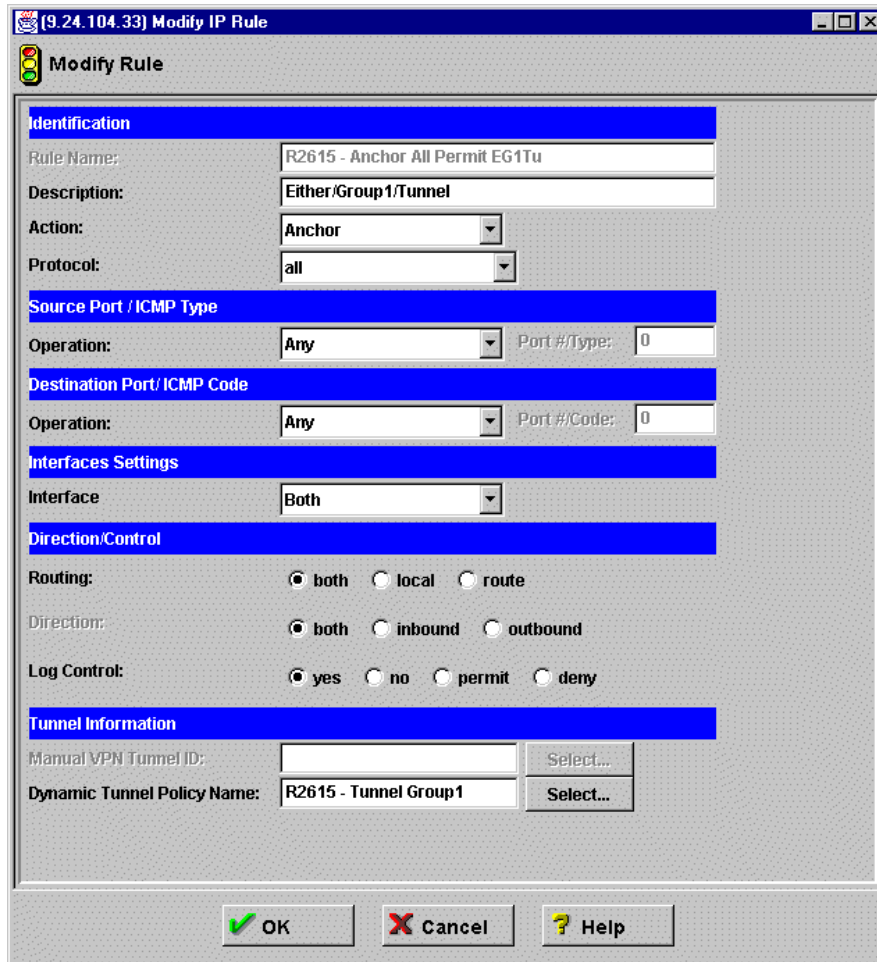


Figure 8-43 z/OS RA03 to NT Firewall-1 anchor rule

This anchor rule is used to define a tunnel in tunnel mode allowing all kinds of traffic. It is used to create the dynamic tunnel between OS/390 RA03 and NT Firewall-1 workstation.

The services

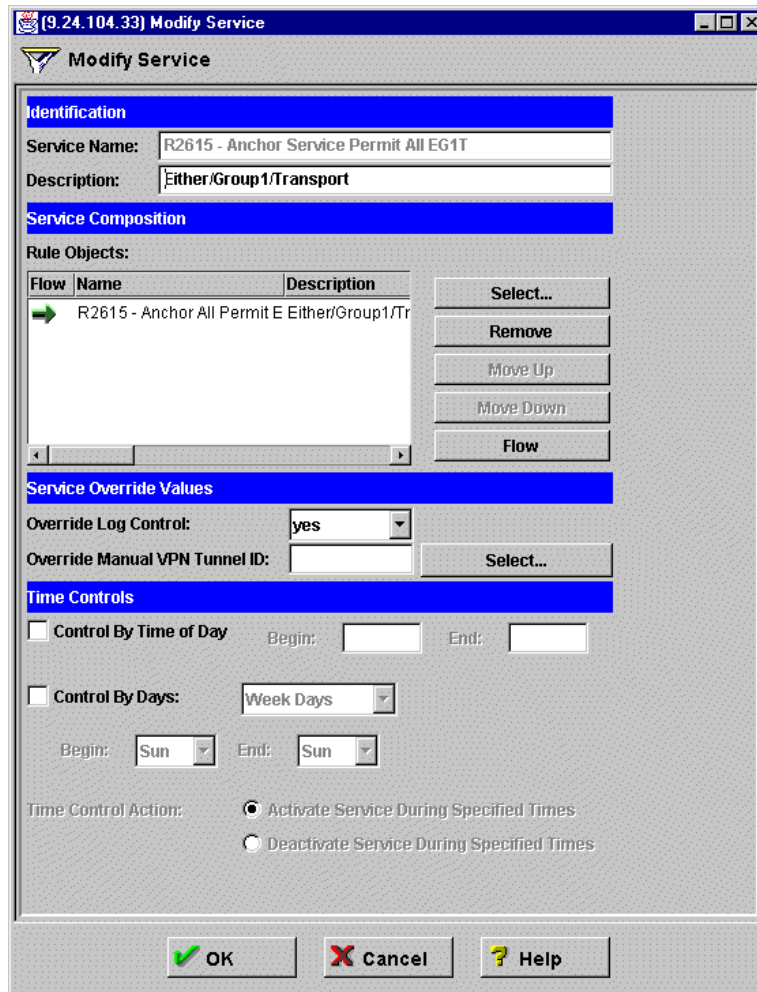


Figure 8-44 z/OS RA03 to z/OS 39 service

This service contains the anchor filter rule to create the connection between the OS/390 RA03 and OS/390 39.

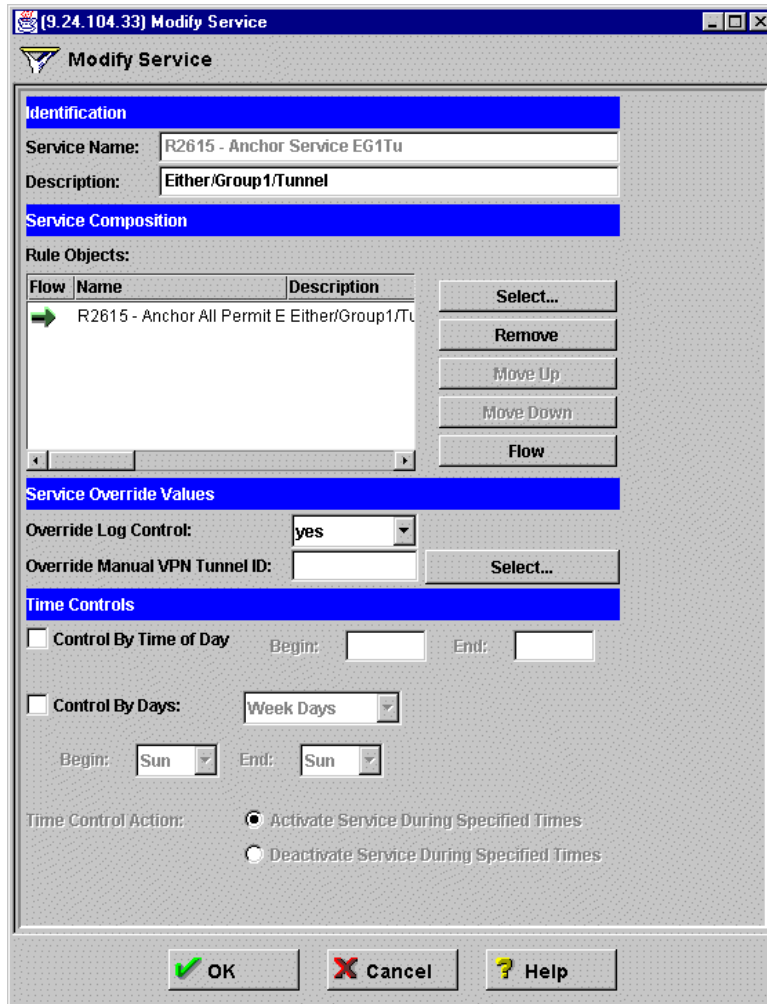


Figure 8-45 z/OS RA03 to NT Firewall-1 service

This service contains the anchor filter rule to create the connection between the OS/390 RA03 and Windows NT Firewall-1 workstation.

The connections

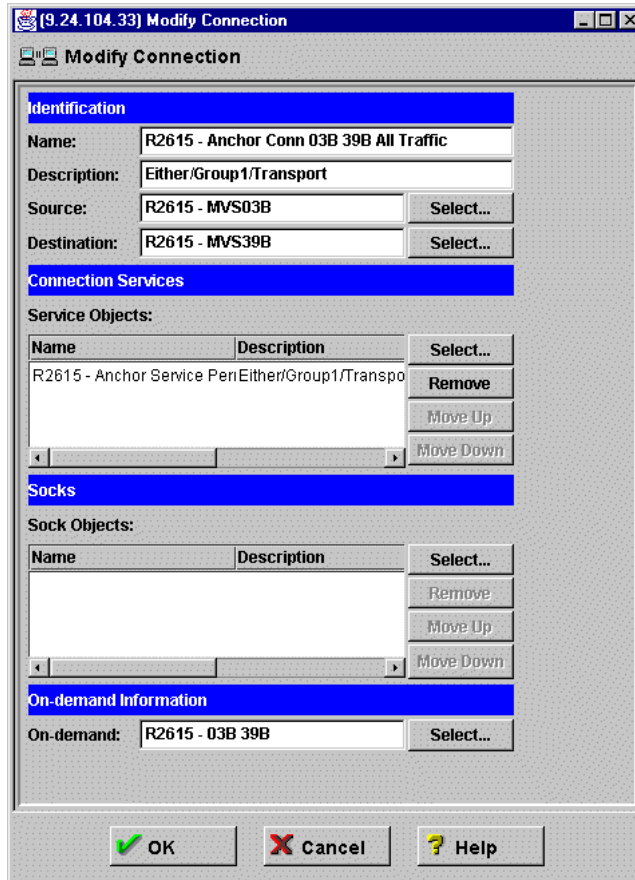


Figure 8-46 z/OS RA03 to z/OS 39 connection

This connection defines the filter rules that are created to represent the tunnel between OS/390 RA03 and OS/390 39. Look at the on-demand definition at the bottom of the window that shows what on-demand object is used if a packet matches the filter rules created by this connection.

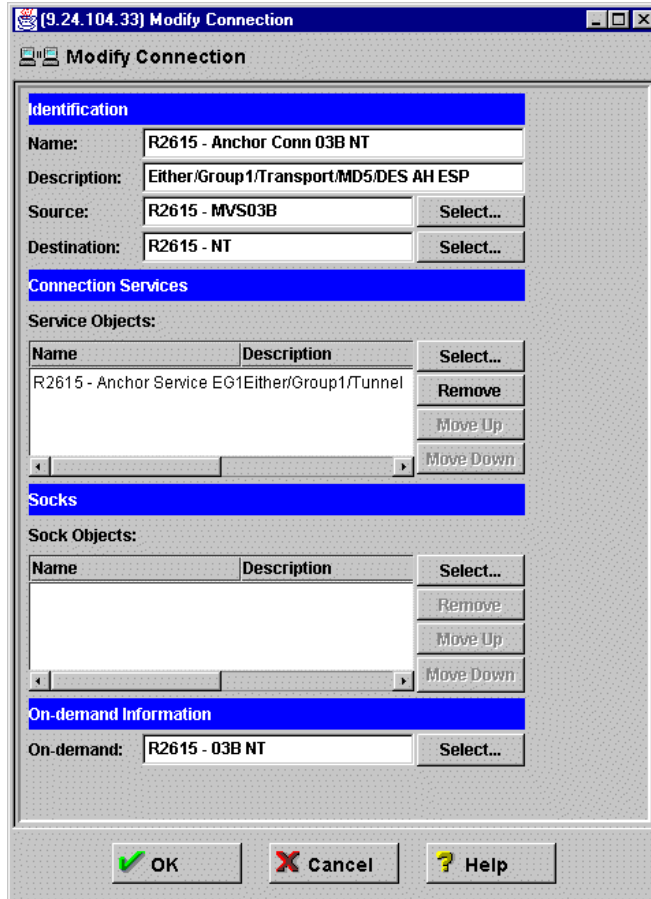


Figure 8-47 z/OS RA03 to NT Firewall-1 connection

This connection defines the filter rules that are created to represent the tunnel between OS/390 RA03 and OS/390 39. Look at the on-demand definition at the bottom of the window that shows what on-demand object is used if a packet matches the filter rules created by this connection.

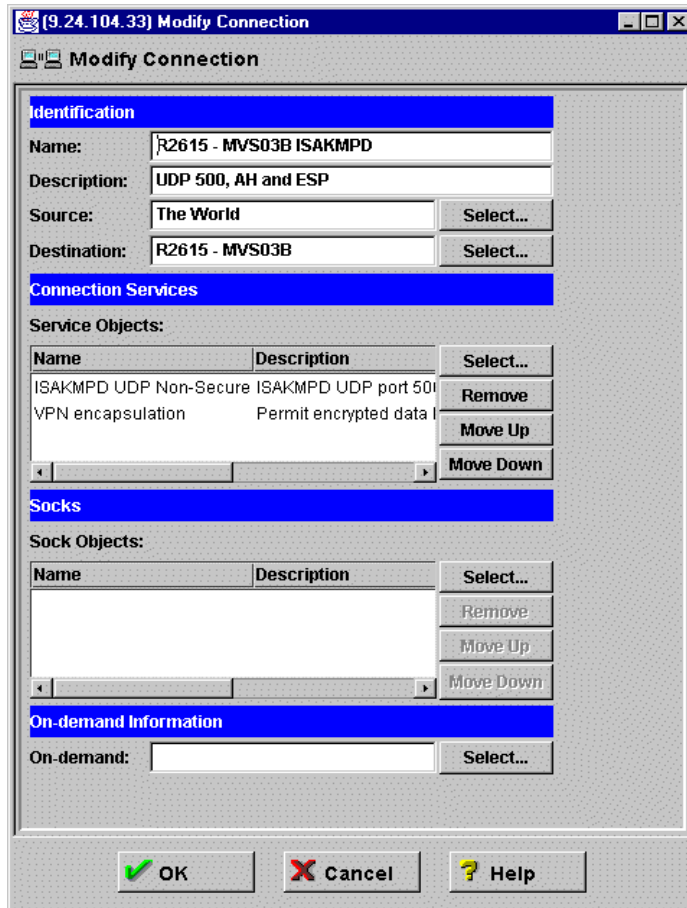


Figure 8-48 z/OS RA03 to the world ISAKMPD protocol connection

This connection allows any IP address to send ISAKMPD and ESP and AH protocols to the OS/390 RA03. You can limit the creation of dynamic tunnels restricting the source address range. We used two default services that come with the firewall to create this connection: the ISAKMPD UDP service and the VPN encapsulation service.

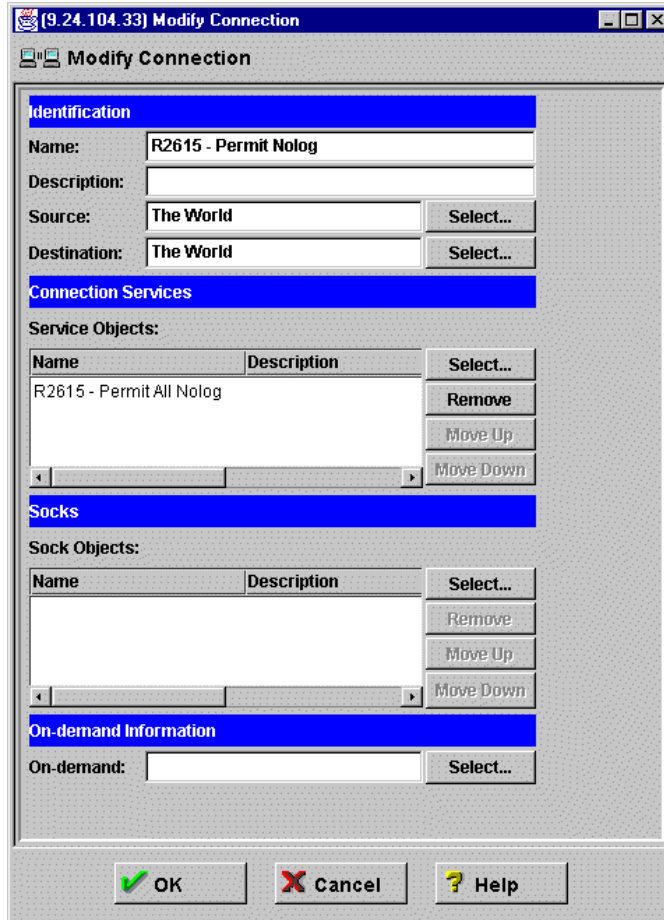


Figure 8-49 z/OS RA03 permit all nolog connection

We created this rule only to facilitate debugging, to prevent the logging of the other traffic in the system syslog.

The connections we created are shown in the following figures.

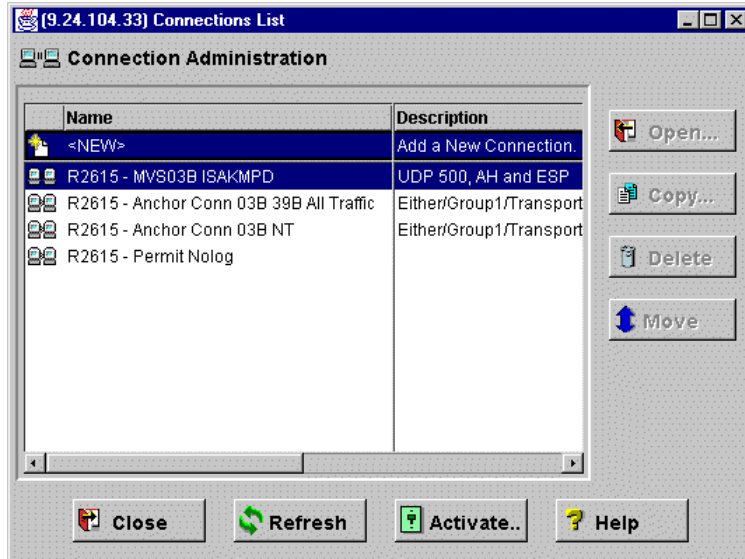


Figure 8-50 FW 03 connection list

The connection order shown above must be followed. The ISAKMPD and VPN encapsulation connection must be defined before any tunnel definition. You have to be sure that the anchor filter rule that defines the dynamic tunnel is selected before any other rule.

Now you can activate the connections you have created by clicking the **Activate** button.

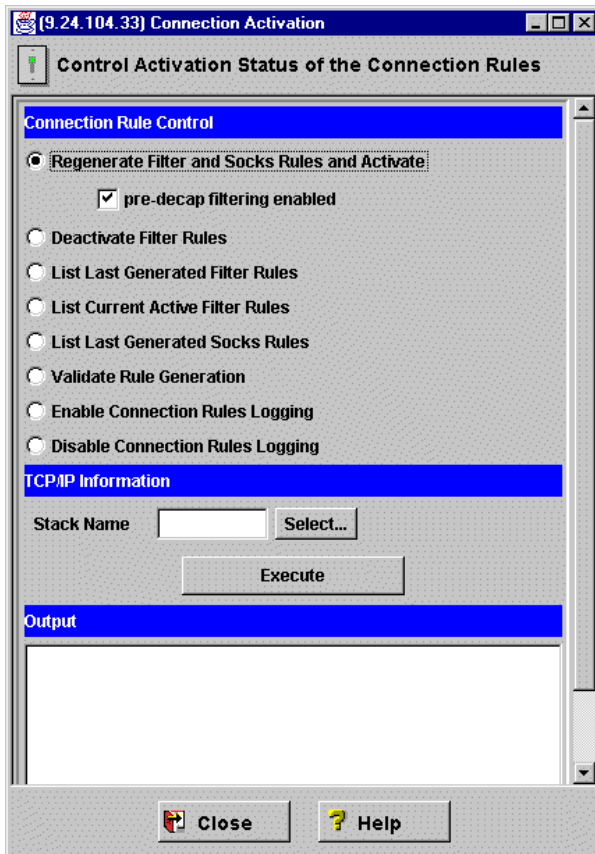


Figure 8-51 FW 03 connection activation

Now check the two first fields and click **Execute**.

Checking **pre-decap filtering enabled** results in AH and ESP packets being filtered before they are decapsulated. If there are concerns about AH and ESP packets in your network, then you may want to have the AH and ESP packets filtered before they are decapsulated.

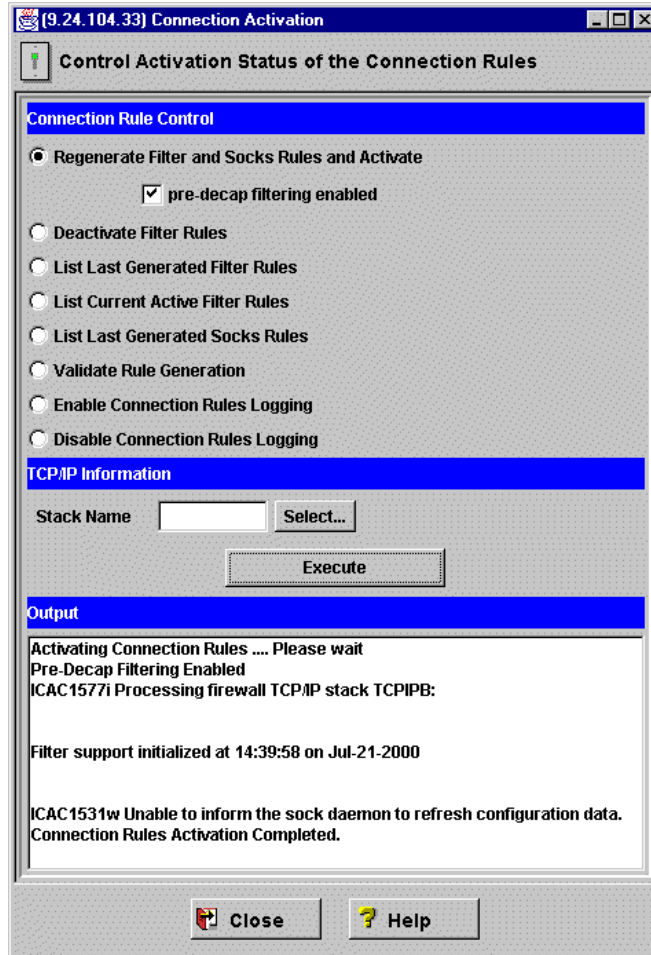


Figure 8-52 FW 03 connection activation messages

On-demand tunnels scenario OS/390 RA39 configuration

We only show the OS/390 RA39 firewall definitions. They are very similar to the OS/390 03.

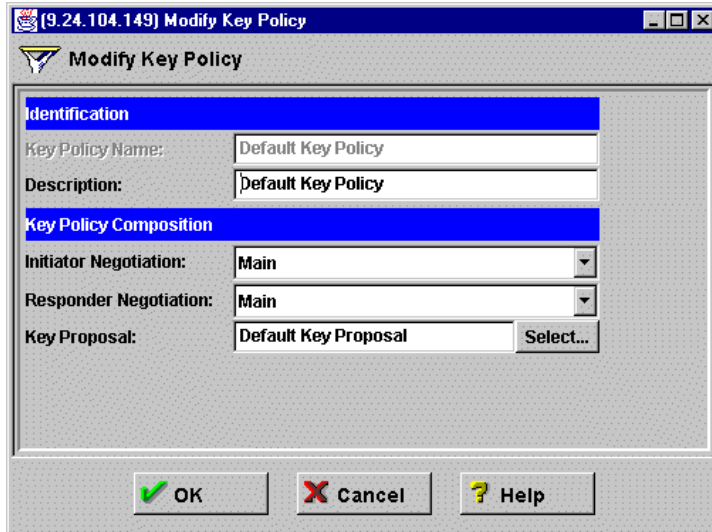


Figure 8-53 FW39 key policy

The OS/390 RA39 uses the same key default key policy.

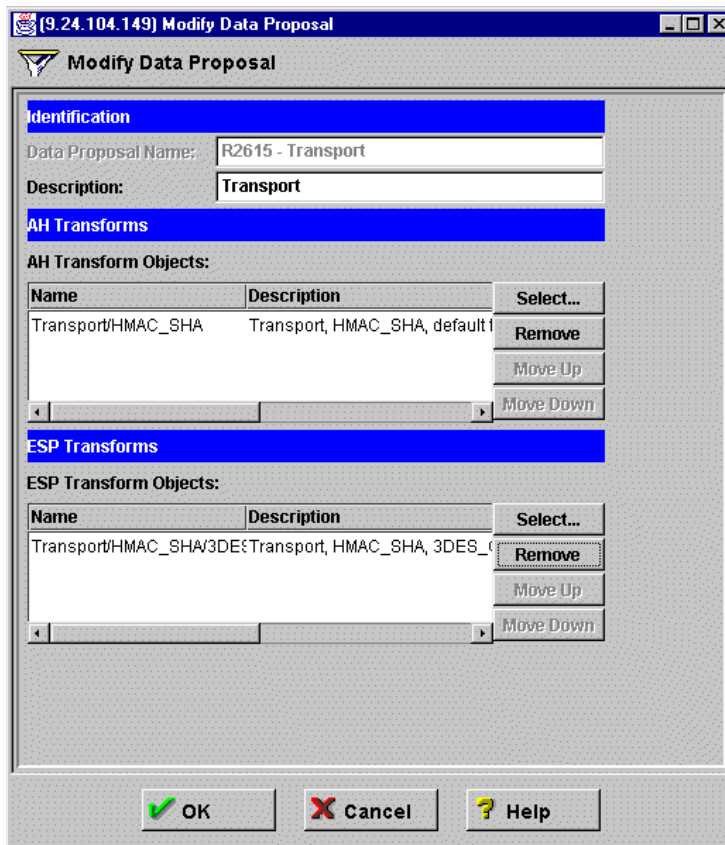


Figure 8-54 FW 39 data proposal

The data proposal contains the same AH and ESP transform objects defined in the OS/390 03 side: transport mode, SHA hashing, and 3DES encryption.

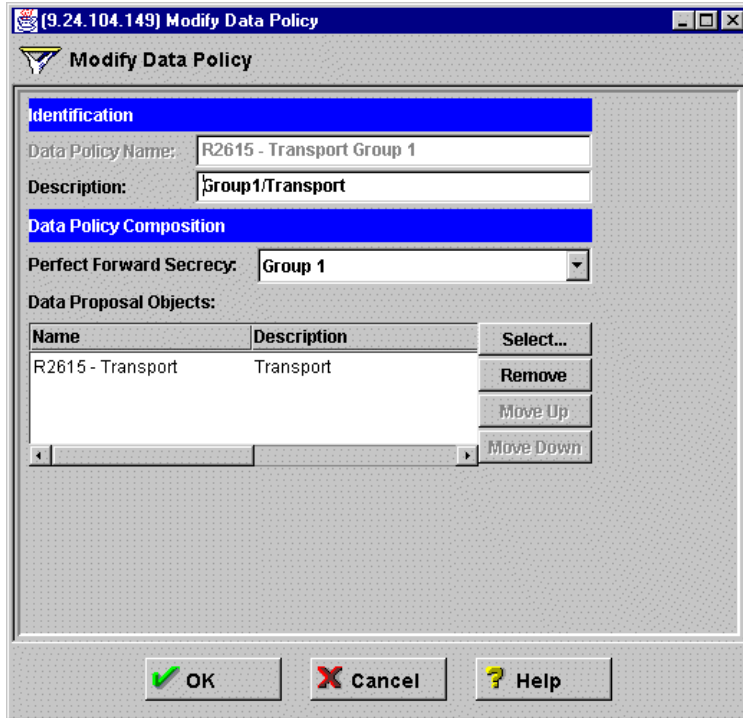


Figure 8-55 FW 39 data policy

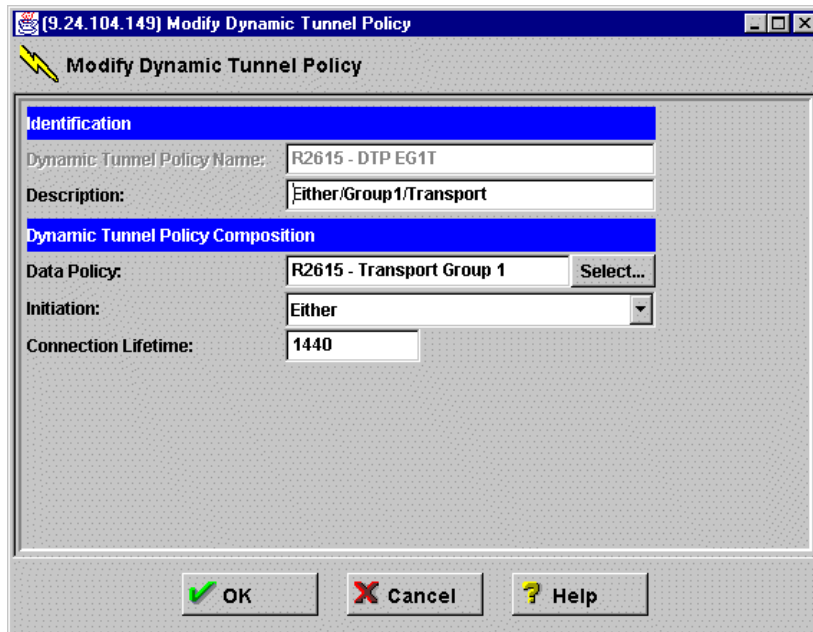


Figure 8-56 FW 39 dynamic tunnel policy

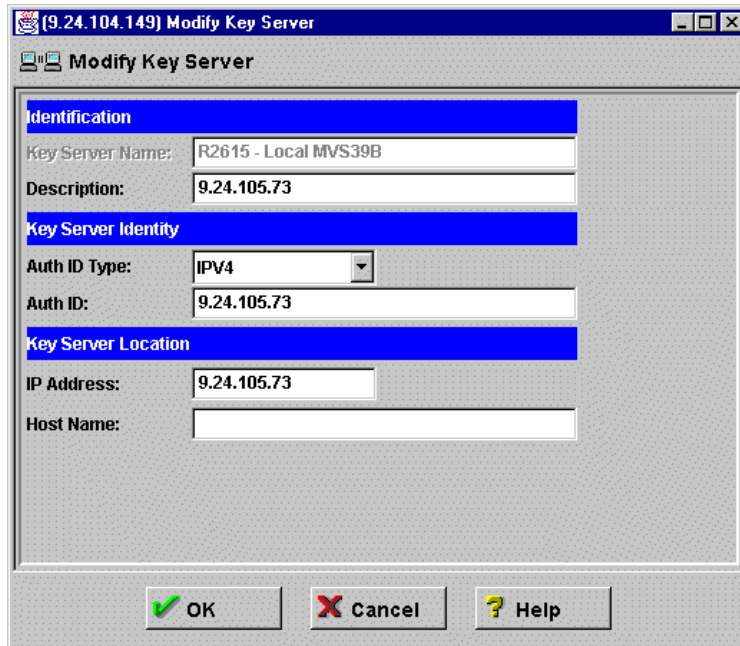


Figure 8-57 FW 39 local key server

This is the local key server.

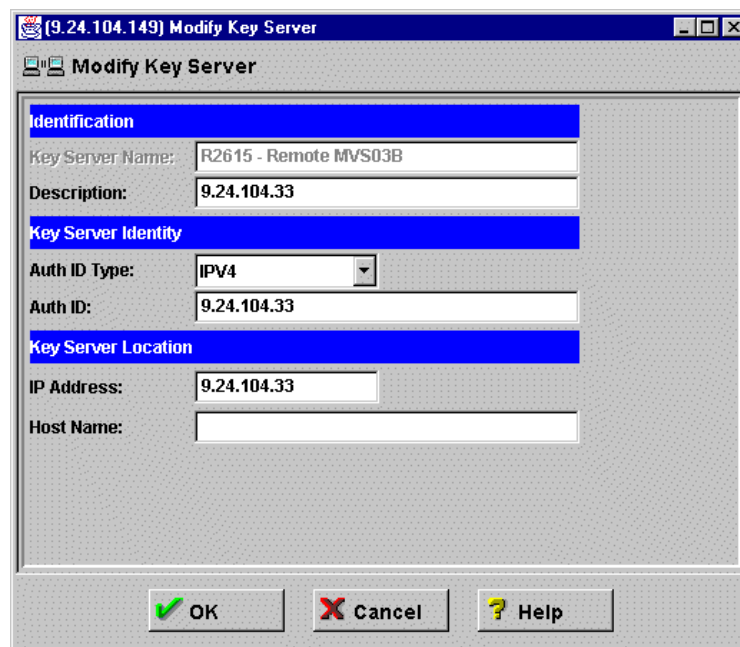


Figure 8-58 FW 39 remote key server

This is the remote key server.

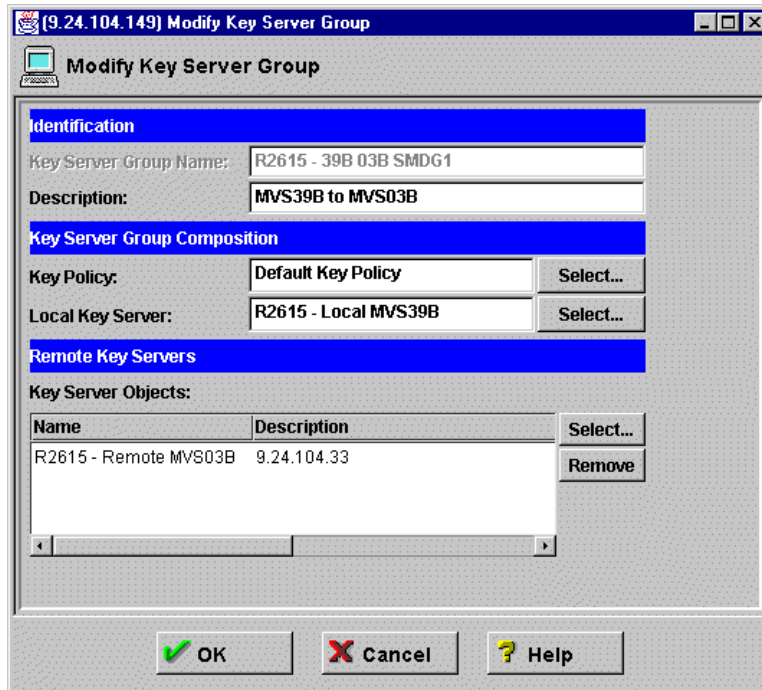


Figure 8-59 FW 39 key server group

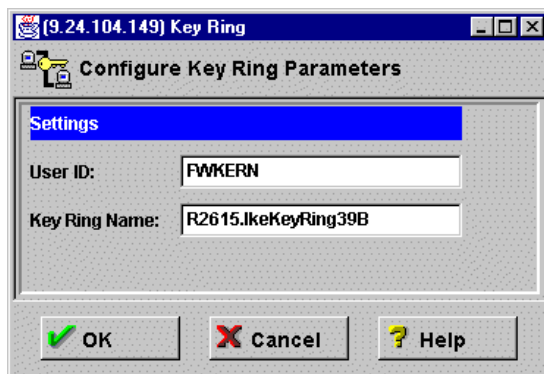


Figure 8-60 FW 39 keyring

This keyring contains a default certificate and a CA certificate for OS/390 03 firewall.

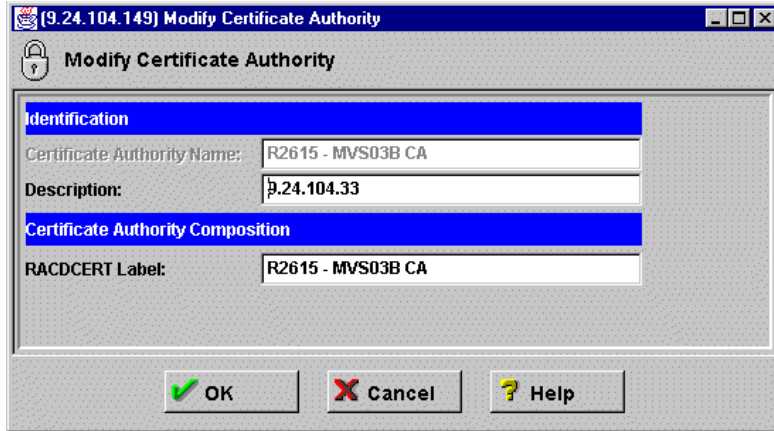


Figure 8-61 FW 39 Certificate Authority

The OS/390 03 CA is shown in Figure 8-62.

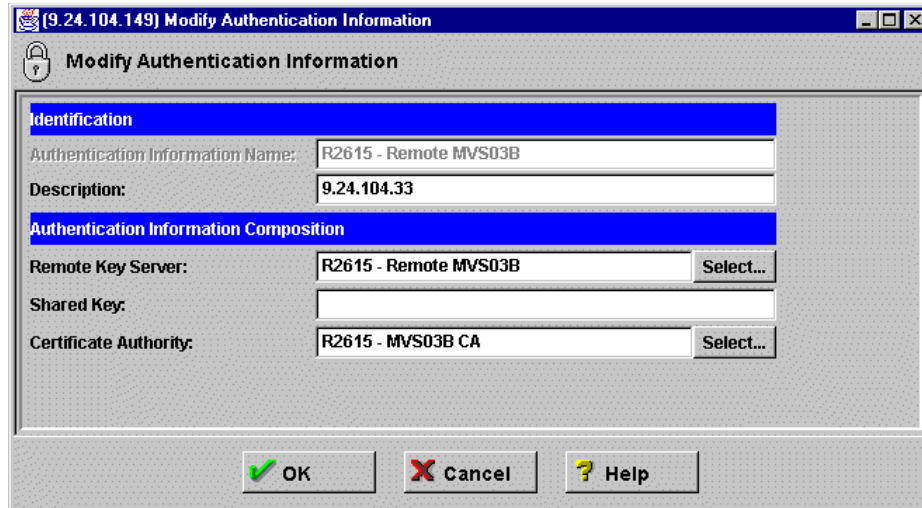


Figure 8-62 FW 39 authentication information

The system 03 is authenticated using the RSASIG method.

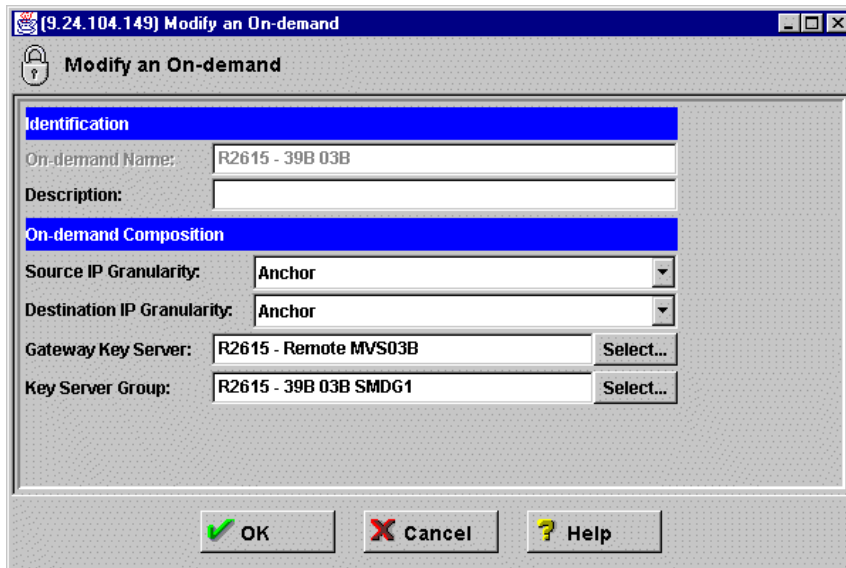


Figure 8-63 FW 39 on-demand object

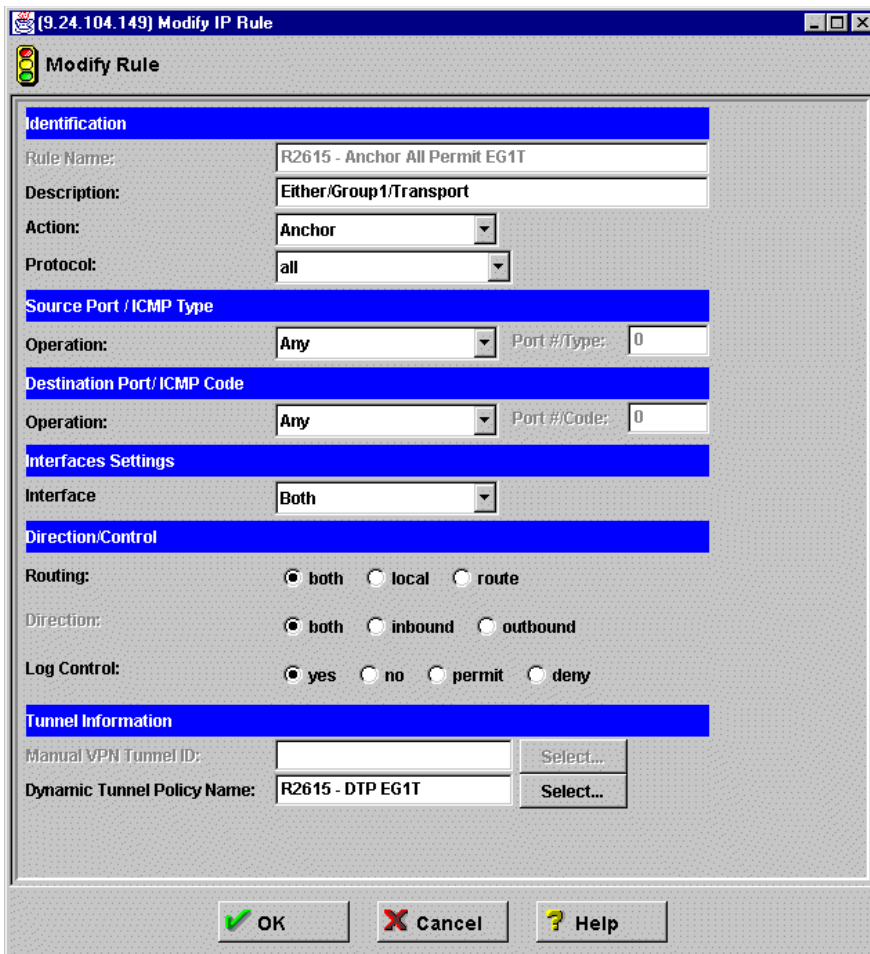


Figure 8-64 FW 39 anchor rule

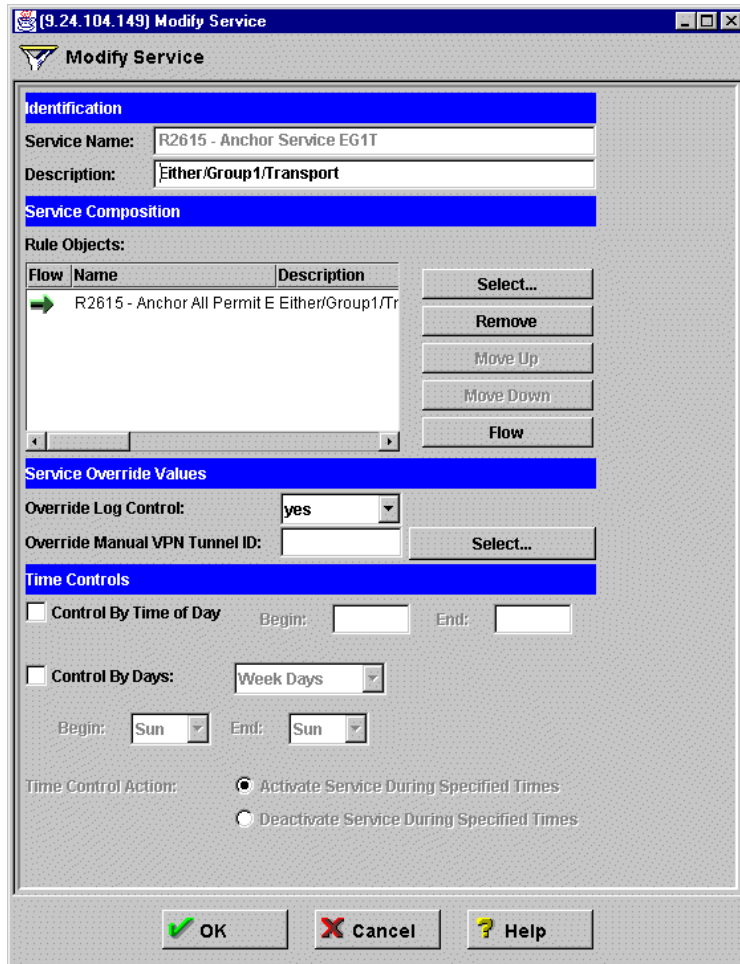


Figure 8-65 FW 39 service

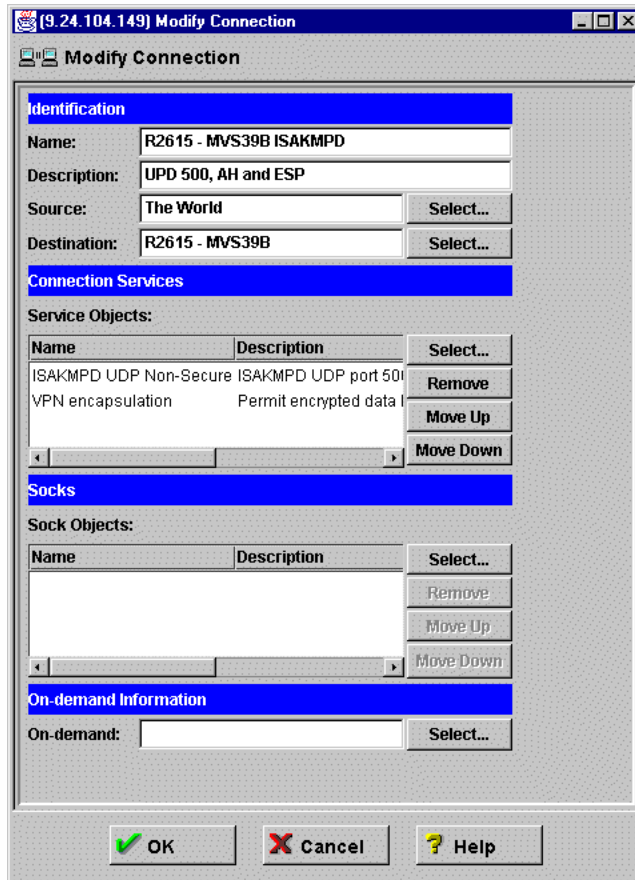


Figure 8-66 FW 39 ISAKMPD, ESP and AH protocols

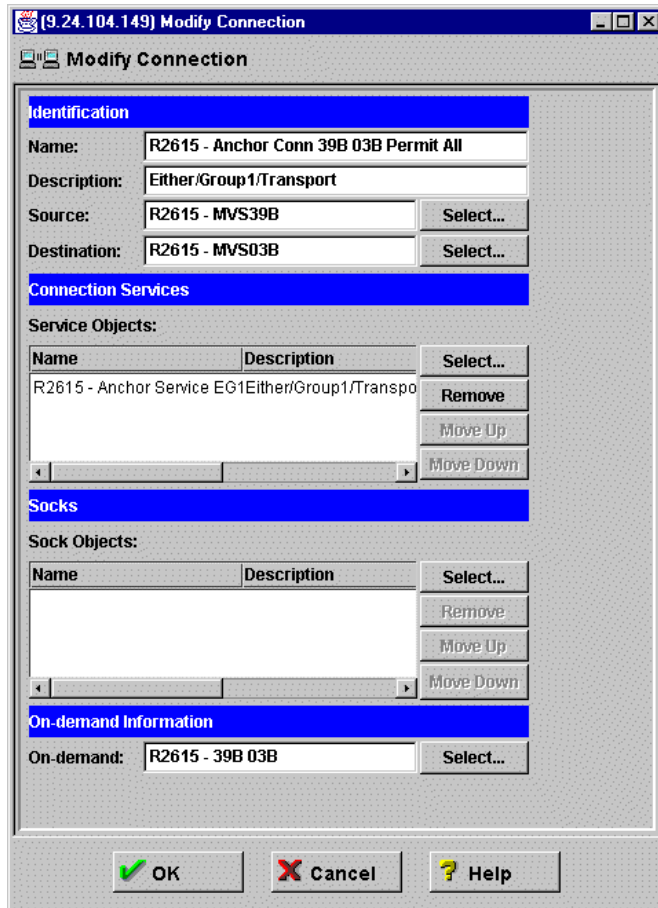


Figure 8-67 FW 39 anchor connection

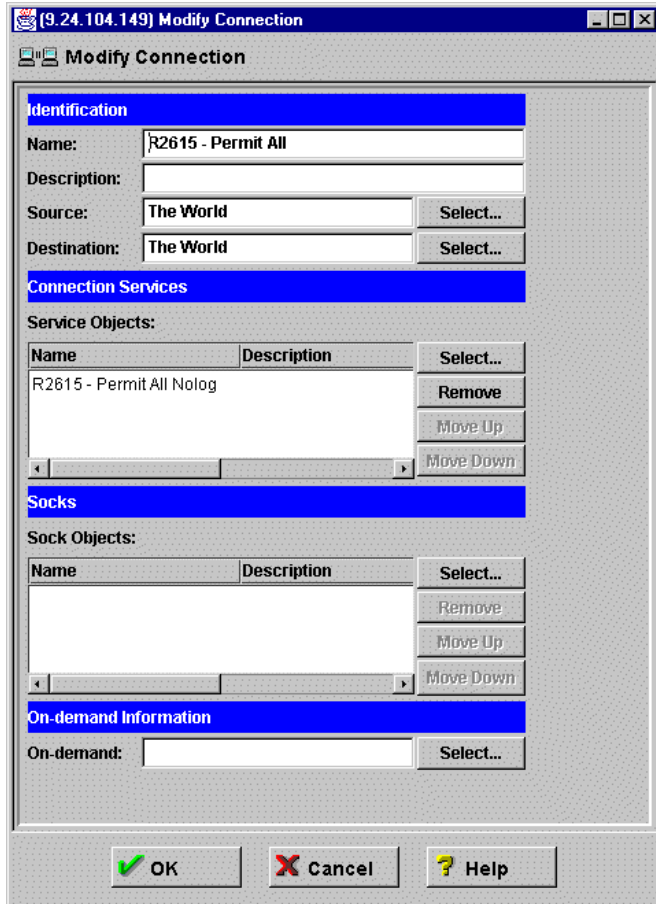


Figure 8-68 FW 39 permit all nolog connection

Now, after you have created all of these objects, you can update the filter rules.

On-demand tunnels scenario NT Firewall-1 configuration

For the Firewall-1, we used the check policy editor to configure the tunnel. We created two rules as shown in Figure 8-69 on page 160.

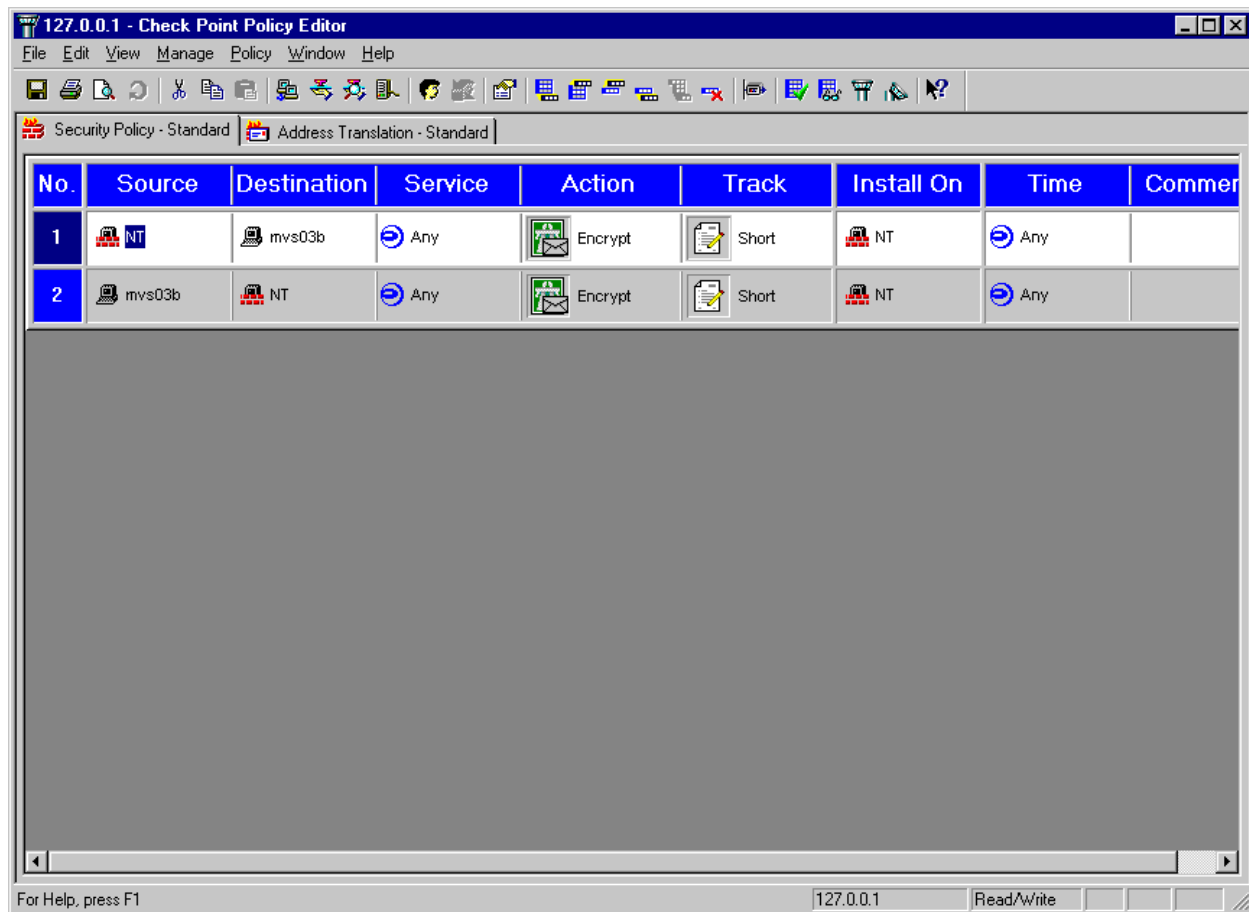


Figure 8-69 FW1 Policy Editor main window

For this tunnel we created only two network objects: the NT and the OS/390 03. The network objects characteristics must follow the tunnel configuration table specifications created earlier in this chapter. The following windows contain the configuration we created.

We created the network objects by selecting **Manage -> Network Objects** .

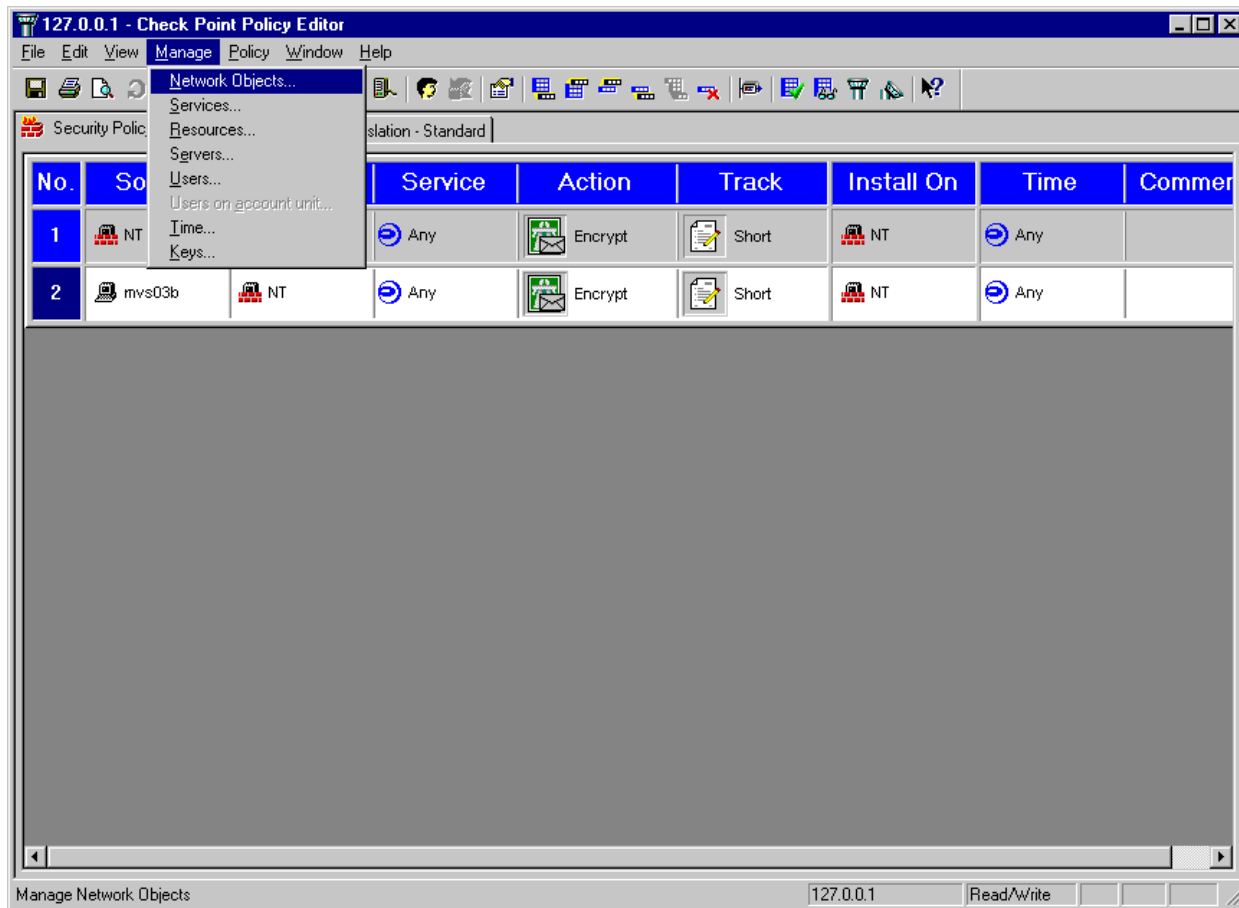


Figure 8-70 FW1 managing network objects

Then we clicked **New** and created the two network objects as shown in Figure 8-71 on page 162.

Creating the NT network object

Fill in the general information about NT.

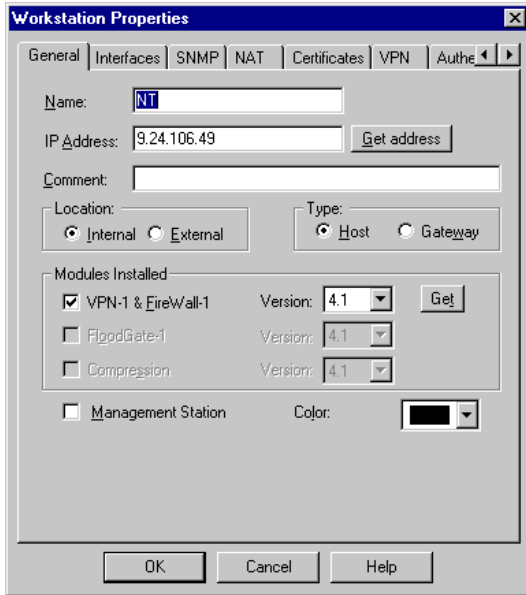


Figure 8-71 FW1 NT general properties

Click the **VPN** tab to configure the VPN options, as shown in Figure 8-72

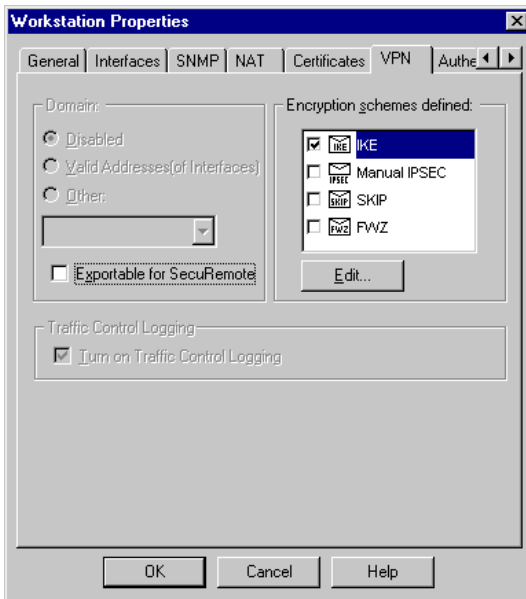


Figure 8-72 FW1 NT VPN properties

Click the **Edit** button and configure the IKE options. These options are used in phase 1 to authenticate the partners and what encryption algorithms, authentications methods, and hashing methods are used for this tunnel:

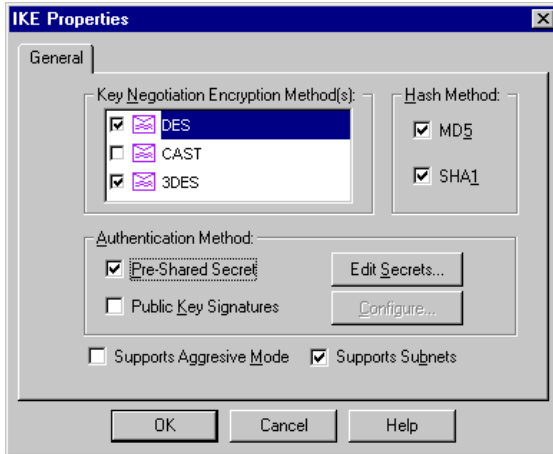


Figure 8-73 FW1 NT IKE properties

Click the **Edit Secrets** button. We have to define the pre-shared key we are using for this tunnel.

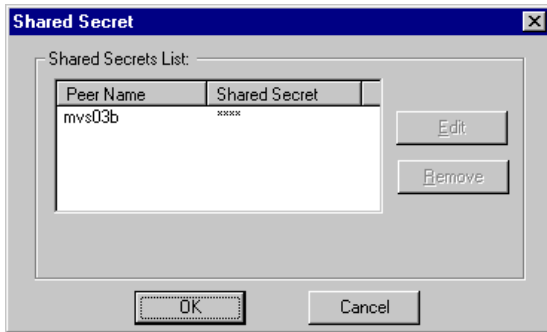


Figure 8-74 FW1 NT MVS03B shared secrets

Double click the **mvs03b** peer name to define the pre-shared key.

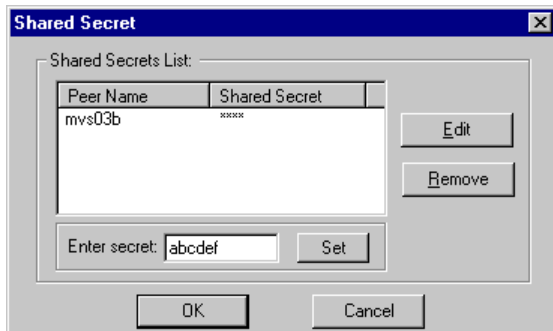


Figure 8-75 FW1 NT MVS03B shared secrets key

Here you have to define the pre-shared key in ASCII format, not in binary format as on the OS/390 firewall. This value must match the value added in Figure 8-39 on page 138 for the authentication information in the OS/390 firewall.

Now repeat the same process for the OS/390 03.

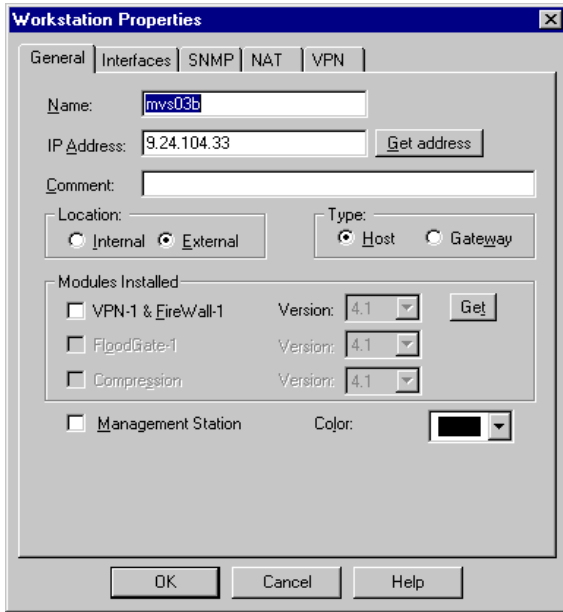


Figure 8-76 FW1 MVS03B general properties

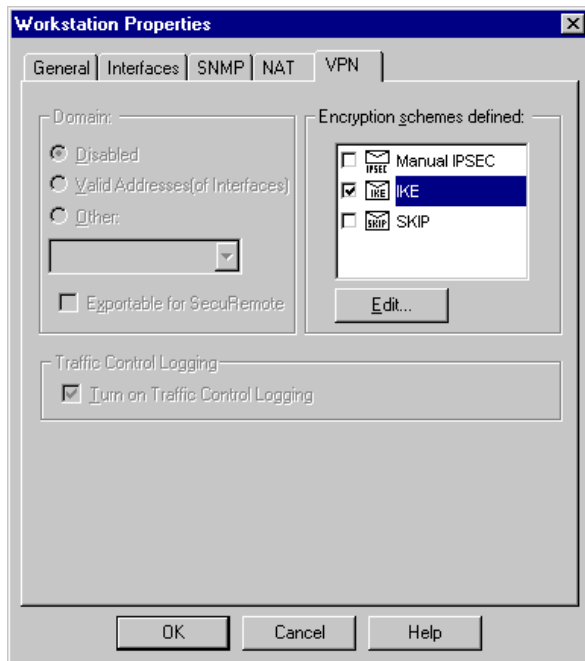


Figure 8-77 FW1 MVS03B VPN properties

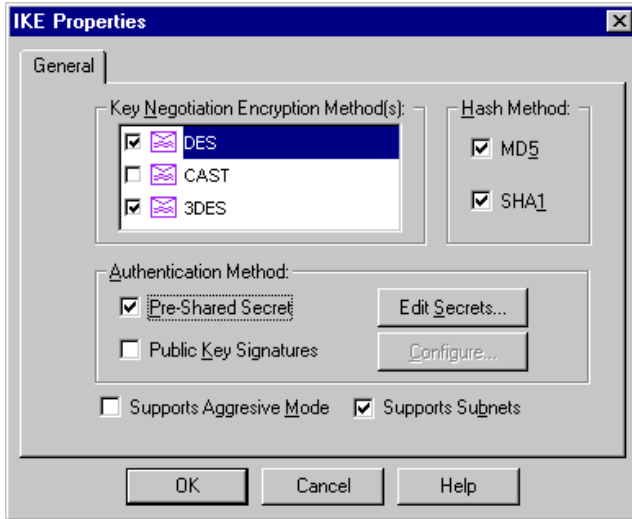


Figure 8-78 FW1 MVS03B IKE properties

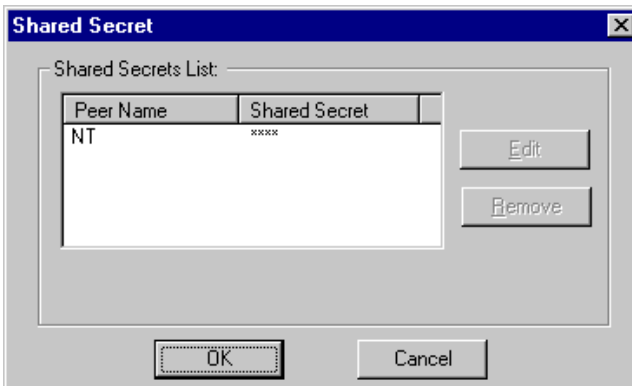


Figure 8-79 FW1 MVS03B shared secrets

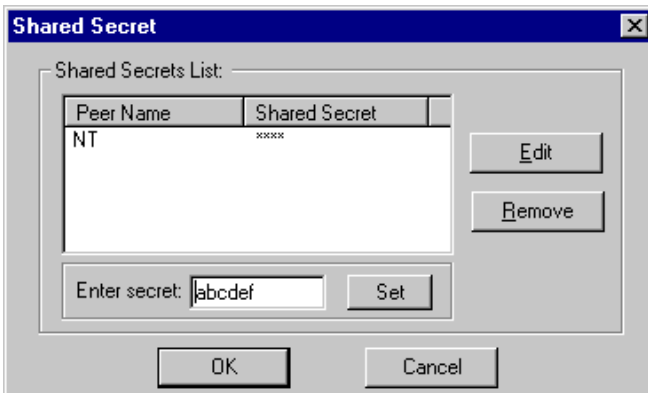


Figure 8-80 FW1 MVS03B shared secrets key

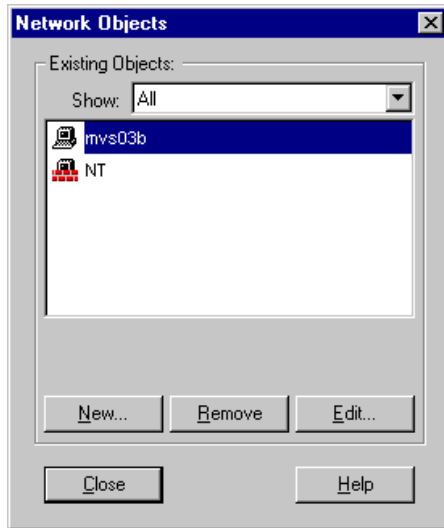


Figure 8-81 FW1 network objects created

The two network objects are created.

Now we have to create two rules: one for the traffic flowing in the NT direction. Both rules permit any kind of traffic, such as the anchor rule in OS/390, and the only action that they have is to encrypt the data. This action also decrypts the data that arrives from OS/390 03.

To create a rule, follow the instructions shown in Figure 8-82 on page 167.

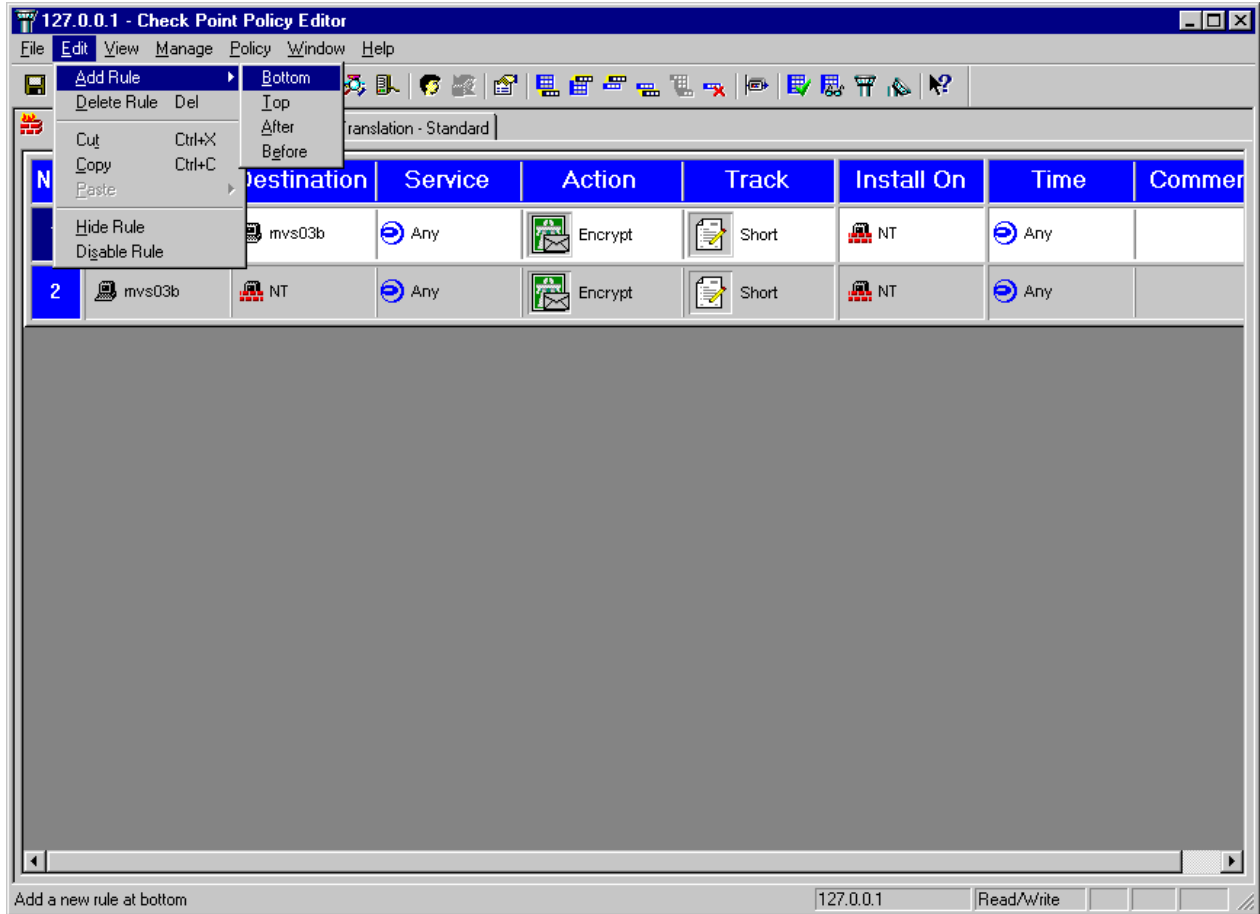


Figure 8-82 FW1 adding the rules

To edit the rule definitions, such as the source and destination, service and action, right-click the field as shown in Figure 8-83 on page 168.

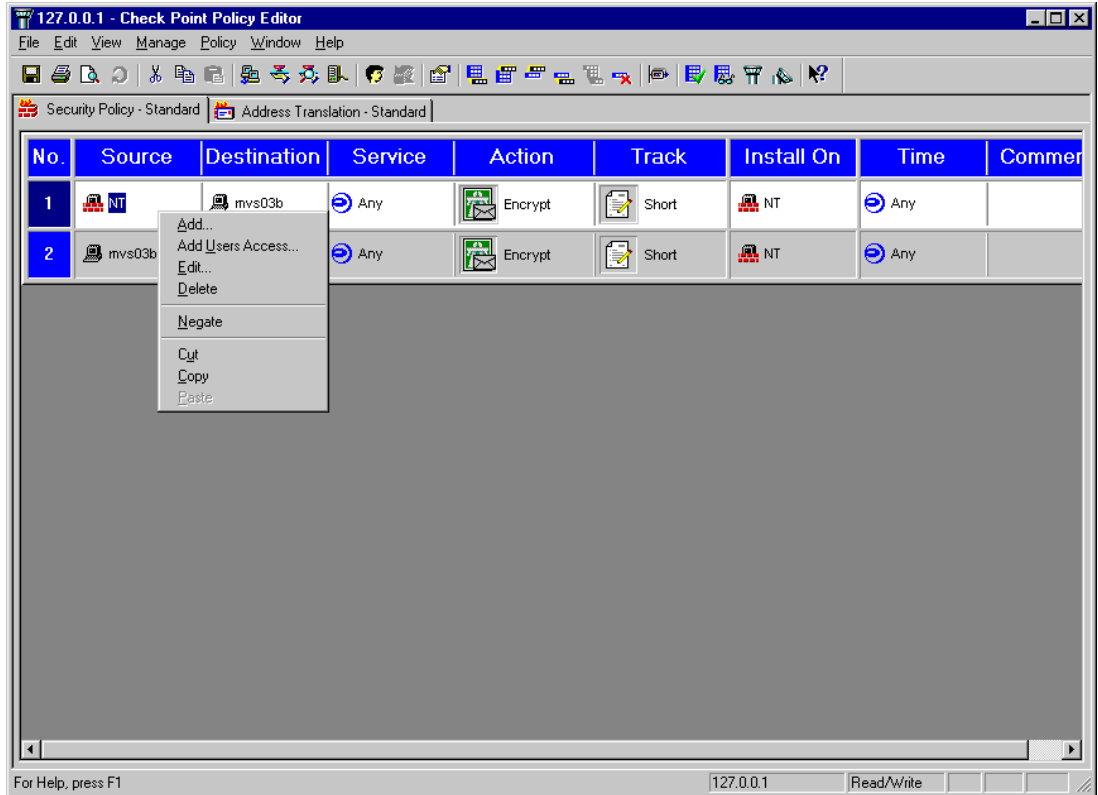


Figure 8-83 FW1 managing the rule definitions

Create the rules as shown in Figure 8-69 on page 160.

Now that we have the rules created, we have to configure the encryption information in the main window. Right-click the action to configure the encryption option.

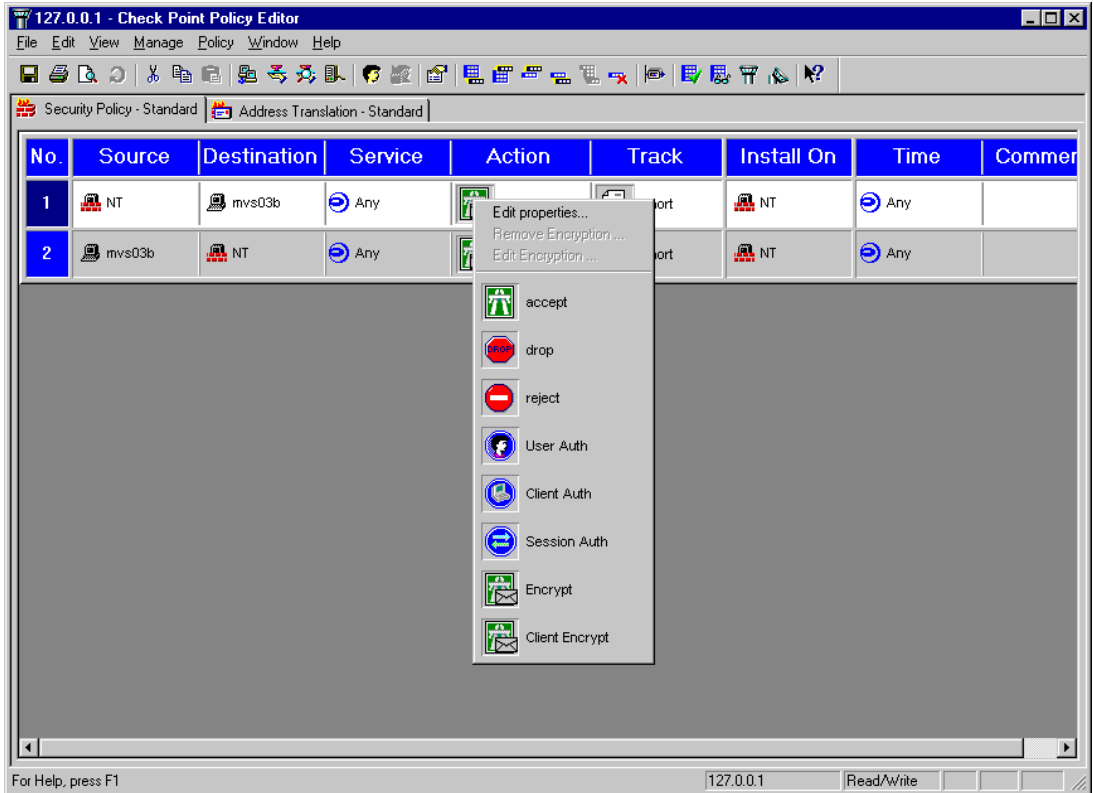


Figure 8-84 FW1 editing the encryption options

First we selected **Encrypt** and then **Edit properties**. The properties are configured like this.

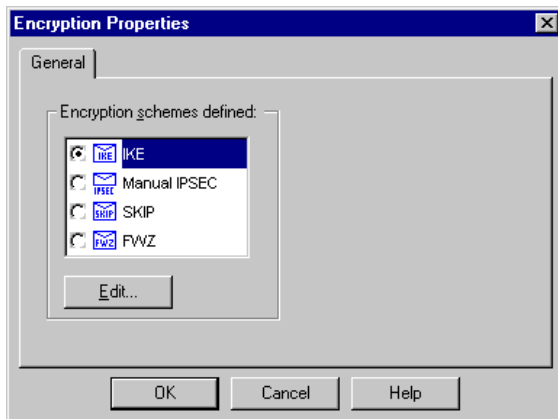


Figure 8-85 FW1 editing the encrypt IKE properties

Select **IKE** and click the **Edit** button.

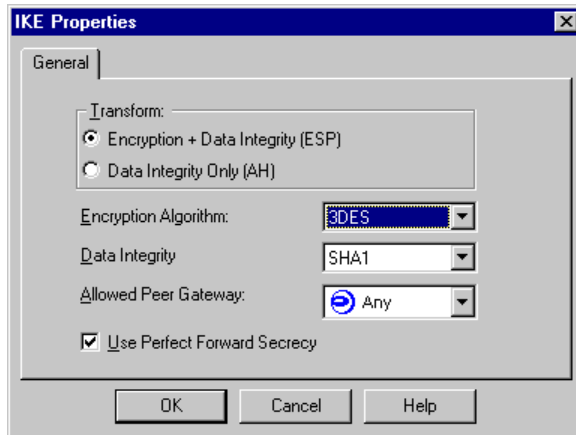


Figure 8-86 Configuring IKE properties

In z/OS, we selected the Perfect Forward Secrecy Group 1 in the data policy for this tunnel. Here we cannot configure Group 1 or Group 2 as in the OS/390 firewall and we cannot select the ESP and AH protocol at the same time. You will not be able to create a tunnel between the Firewall-1 and OS/390 firewall using both protocols. You can have only the ESP or only the AH protocol, but not both at same time.

These options are used in phase 2 of the tunnel negotiation and must match the options specified in the OS/390 03 dynamic tunnel policy options.

The default value for the renegotiated IKE security association in the Firewall-1 is bigger than the default and maximum value for the OS/390 firewall. The security association is not created if you don't change this value. You can change this value using the windows in the following sequence.

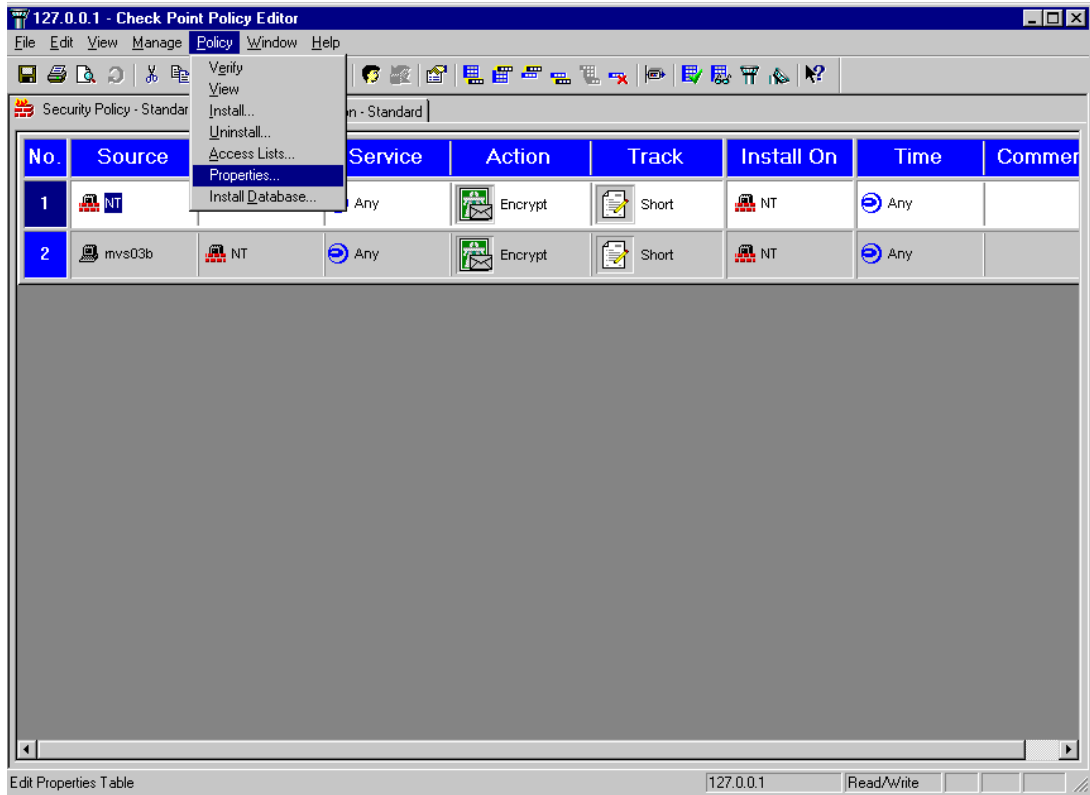


Figure 8-87 FW1 policy properties

Select **Policy** -> **Properties**.

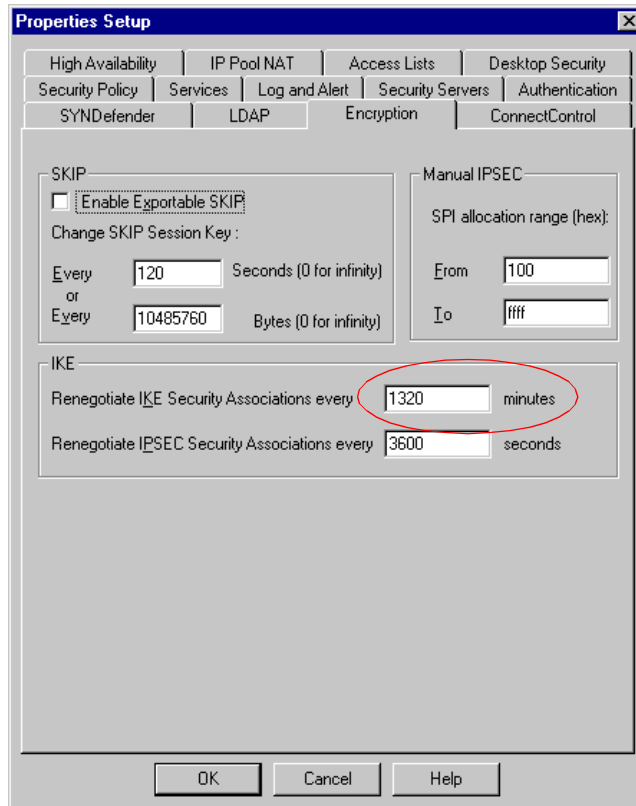


Figure 8-88 FW1 policy encryption properties

The maximum value for the z/OS firewall is 86400 seconds (one day). Be sure that the value of this field is less than or equal to 86400 seconds. The value here is expressed in minutes.

Now you can install the policy and start the tunnel. The on-demand tunnels are dynamically created by the z/OS firewall when a packet arrives at the interface and matches a filter rule associated to an on-demand object.

To install the policy follow the instructions below.

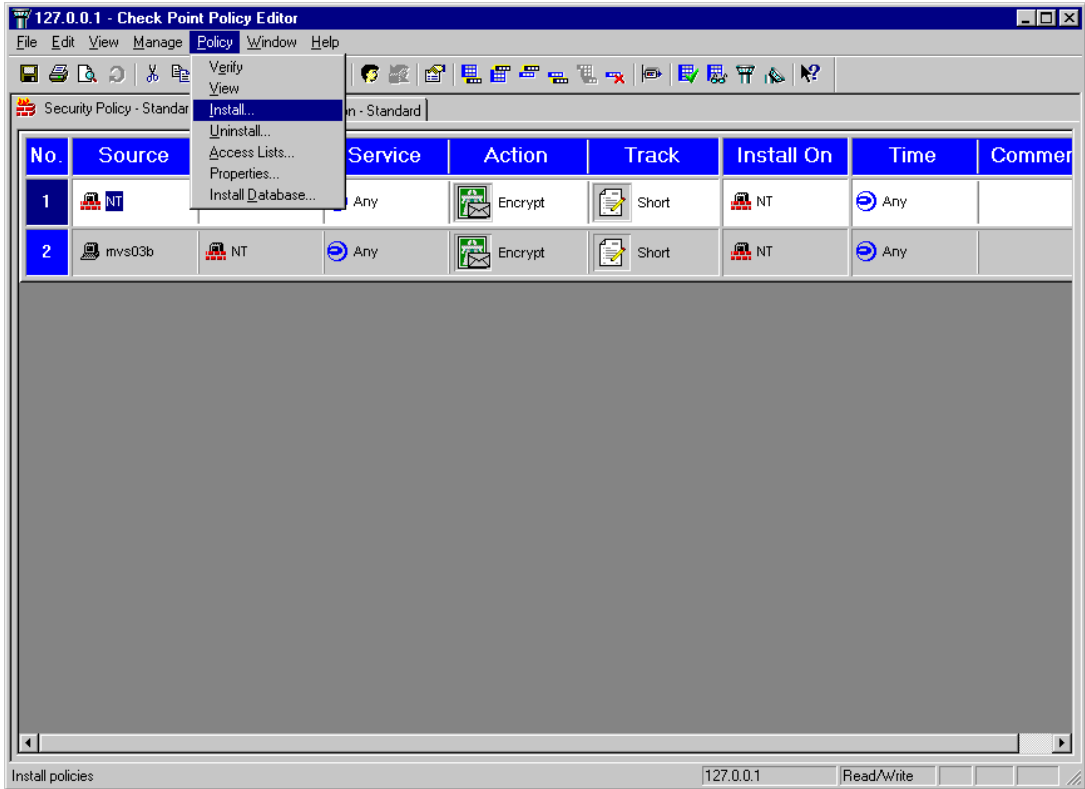


Figure 8-89 FW1 installing the policy menu

Select **Policy -> Install**. This updates the Firewall-1 filter rules.

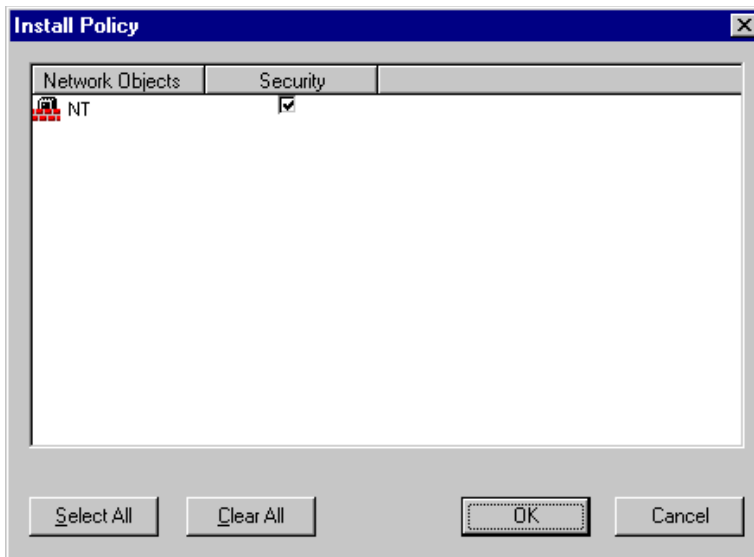


Figure 8-90 FW1 selecting the targets

Click the **OK** button.

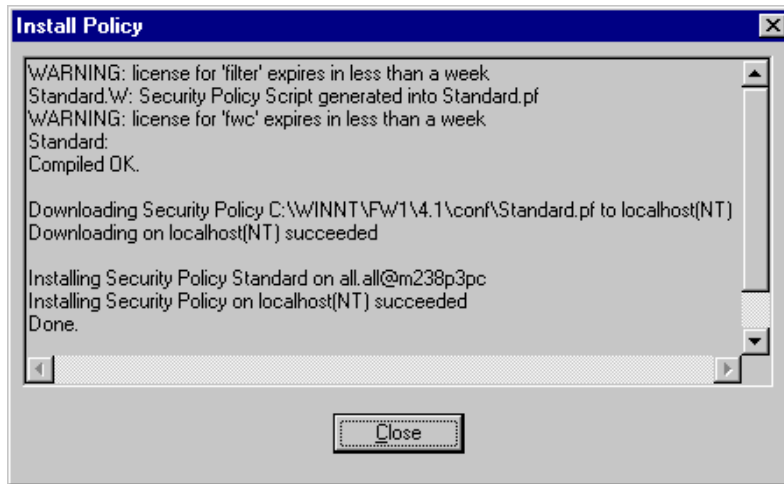


Figure 8-91 FW1 policy installed

Now the policy you created is installed.

You can start the tunnel by just issuing a ping to the OS/390 03 machine.

Be sure that this ping matches the rule you created. There is an option in the Firewall-1 properties you have to check before using a ping to create the connection. If you want, you can use any other TCP/IP application to start the tunnel. Check the properties again on this window.

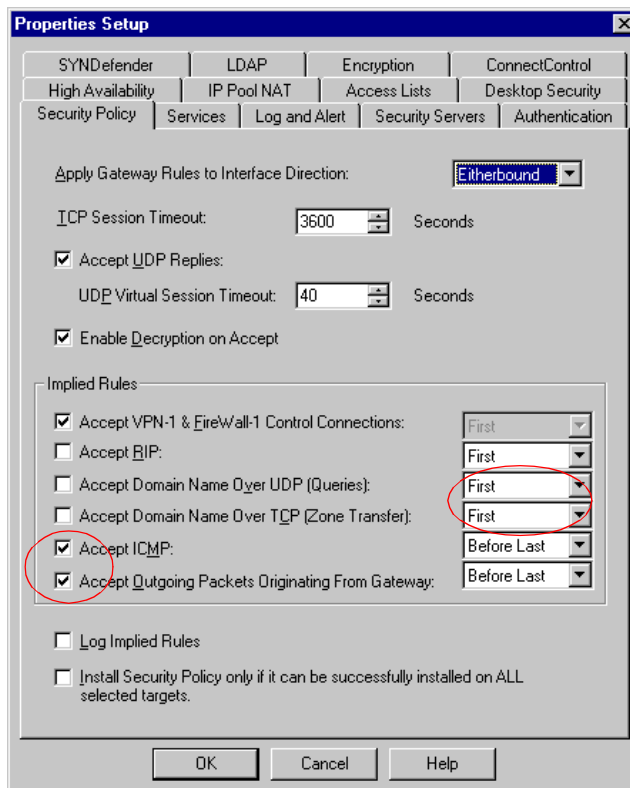


Figure 8-92 FW1 security policy properties

Be sure that the ICMP packet matches the first rule that controls the traffic between NT and OS/390 03.

```
C:\>ping 9.24.104.33

Pinging 9.24.104.33 with 32 bytes of data:

Request timed out.
Reply from 9.24.104.33: bytes=32 time<10ms TTL=64
Reply from 9.24.104.33: bytes=32 time<10ms TTL=64
Reply from 9.24.104.33: bytes=32 time=10ms TTL=64
```

The attempts time out until the tunnel is created. After that they will work. In the ISAKMPD job log messages, you see the following messages.

```
ICA8296i: ISAKMP security association 0000000001 created from 9.24.104.33 to
9.24.106.49 exchange type:MAIN auth method:SHAREDKEY encralg:3DES_CBC hashalg:
SHA DHGroup:GROUP2 lifetime:79200 lifesize:NONE.
ICA8227i: Dynamic tunnel 0000000002 created from 9.24.104.33:255.255.255.255
protocol:ALL port:ALL to 9.24.106.49:255.255.255.255 protocol:ALL port:ALL
encralg:3DES_CBC encr_authalg:HMAC_SHA encr_mode:TUNNEL authalg:NONE auth_mode:
NONE SA lifetime:3600 SA lifesize:NONE connection lifetime:86400 PFS:GROUP1.
```

All the following traffic flows between OS/390 03 and Windows NT Firewall-1 workstation using this tunnel. You can check the firewall log at the syslogd daemon log to check the messages.

```
ICA1073i;TCPIPB;R;p;i;;9.24.104.33;s;;9.24.106.49;d;;9.24.104.33;p;;icmp;t;;
8;c;;0;r;;l;a;n;f;;n;T;;0000000503:0000000502:0000000503:0000000502:0000000501:
0000000503:0000000503:0000000504:0000000002;AH;;0;ESP;;0;l;;60;
ICA1073i;TCPIPB;R;p;o;;9.24.104.33;s;;9.24.104.33;d;;9.24.106.49;p;;icmp;t;;
0;c;;0;r;;l;a;n;f;;n;T;;0000000503:0000000502:0000000503:0000000502:0000000501:
0000000503:0000000503:0000000504:0000000002;AH;;0;ESP;;0;l;;60;
```

These two firewall log messages show two ICMP packets, one inbound and another outbound, flowing through the tunnel ID 2.

One interesting thing can happen: if you try to deactivate the tunnel using the GUI interface on the other side of the tunnel, the tunnel is automatically recreated. The first message flows to tell the ISAKMPD tunnel to be deactivated. When the ISAKMPD answers this message, the firewall intercepts it and a new tunnel is created because there is an anchor rule for it that matches the answer.

You can check the tunnel status by using the GUI interface. Check Figure 8-93 on page 176.

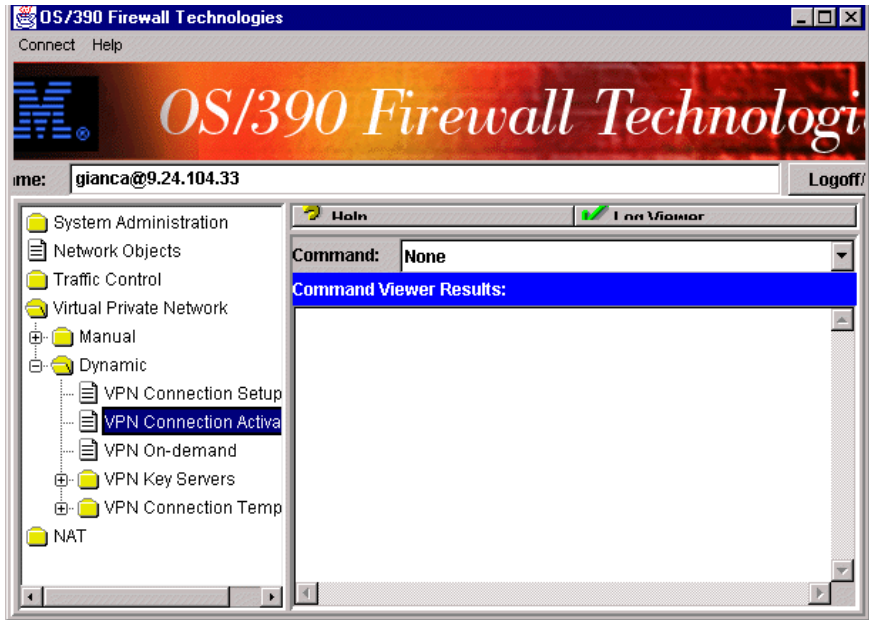


Figure 8-93 FW checking the VPN connections

Double-click **VPN Connection Activation** to see the activated connections.

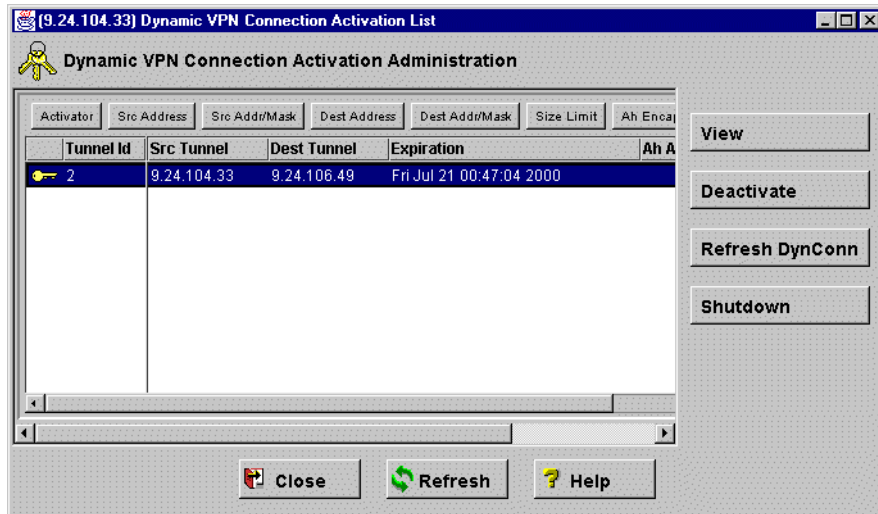


Figure 8-94 FW activated VPN connections

Select the connection and click **View**.

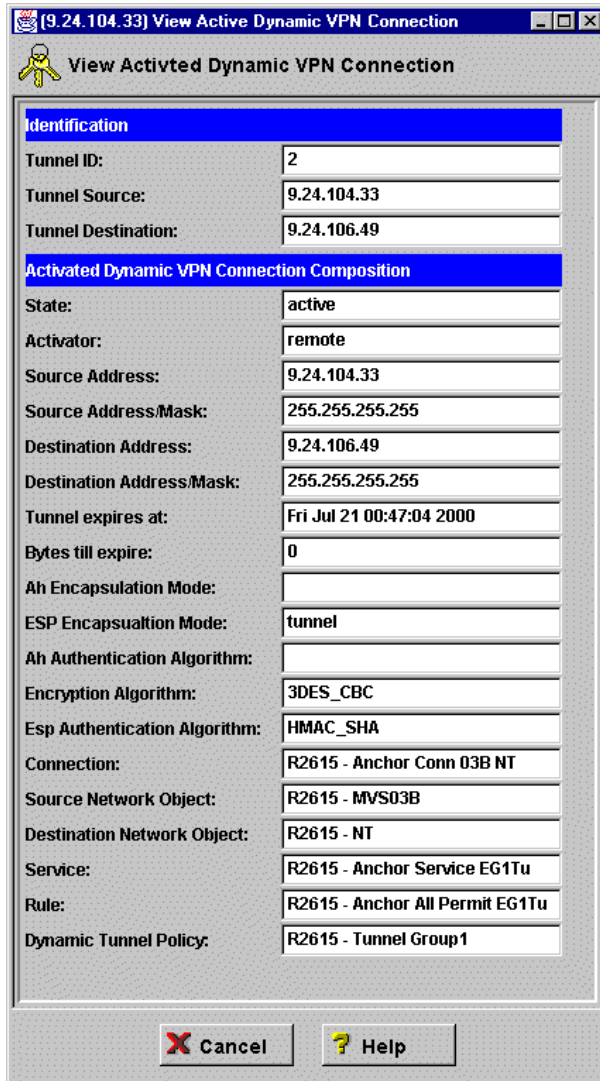


Figure 8-95 On-demand tunnel attributes

As you can see, the tunnel was activated by a remote partner peer. If you start the tunnel by issuing a ping from OS/390 03 to the NT machine, you see a window similar to the one shown in Figure 8-96 on page 178.

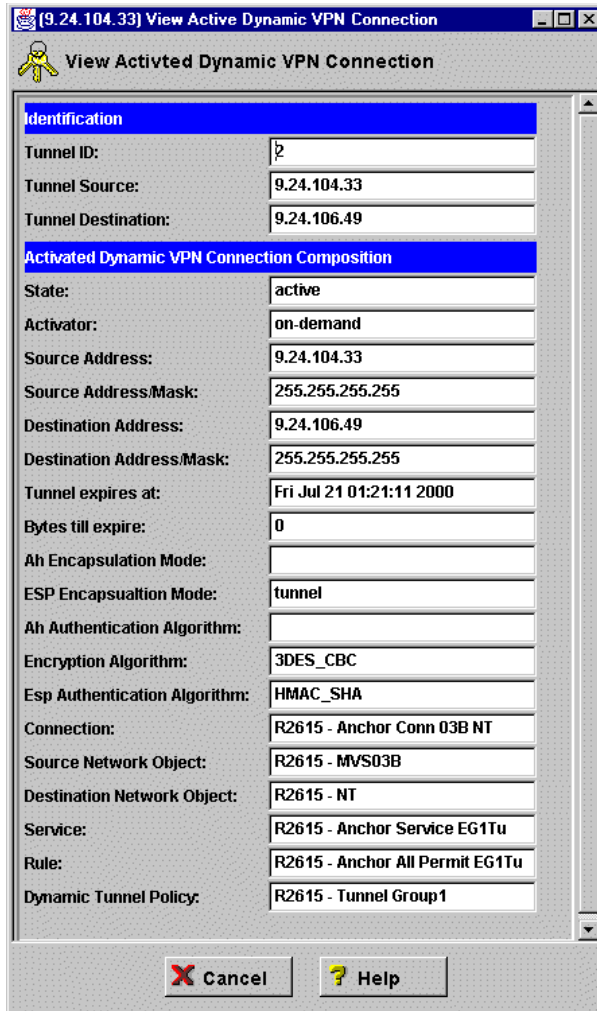


Figure 8-96 On-demand activated tunnel attributes

Now the activation field show that this tunnel was created on-demand. The OS/390 firewall intercepts the packet before sending it to the partner and starts the activation of a dynamic tunnel with the packet destination. The dynamic tunnel is created based on the anchor or packet, depending on how you defined the on-demand object.

The ISAKMPD messages is like this.

```
ICA8298i: Attempting to create on demand connection from 9.24.104.33 to 9.24.106.49. 1
ICA8296i: ISAKMP security association 000000001 created from 9.24.104.33 to 9.24.106.49
exchange type:MAIN auth method:SHAREDKEY encralg:3DES_CBC hashalg:MD5 DHGroup:GROUP1
lifetime:28800 lifesize:NONE.
ICA8227i: Dynamic tunnel 000000002 created from 9.24.104.33:255.255.255.255
protocol:ALL port:ALL to 9.24.106.49:255.255.255.255 protocol:ALL port:ALL
encralg:3DES_CBC encr_authalg:HMAC_MD5 encr_mode:TUNNEL authalg:NONE auth_mode:NONE SA
lifetime:3600 SA lifesize:NONE connection lifetime:86400 PFS:GROUP1. 2
e:3600 SA lifesize:NONE connection lifetime:86400 PFS:GROUP1.
```

1 This message is new because the message that originates the connection was originated from the OS/390 side of the tunnel. When a message comes from the other side, the partner starts phase 1 before sending the message. If you are sending ICMP or UDP, the messages are lost until the tunnel is created. But if you are sending TCP messages, the retransmission time-out algorithm can complete the connection if the tunnel is created on time.

2 As we have mentioned before, we are using the default time for phase 2. If you check this message when the tunnel was created by the Firewall-1, you see a different value. It cannot be larger than this because this is the maximum value that the OS/390 Firewall supports.

Now we can do the same process for the tunnel between the two OS/390s. You can start the tunnel on the 03 side, look at the messages and dynamic connection status, stop the tunnel, and start the tunnel again but from the RA39 side. The windows and messages below were obtained by starting the tunnel from the 03 side.

To start this tunnel we had only to issue a **ping** command to the other side of the tunnel.

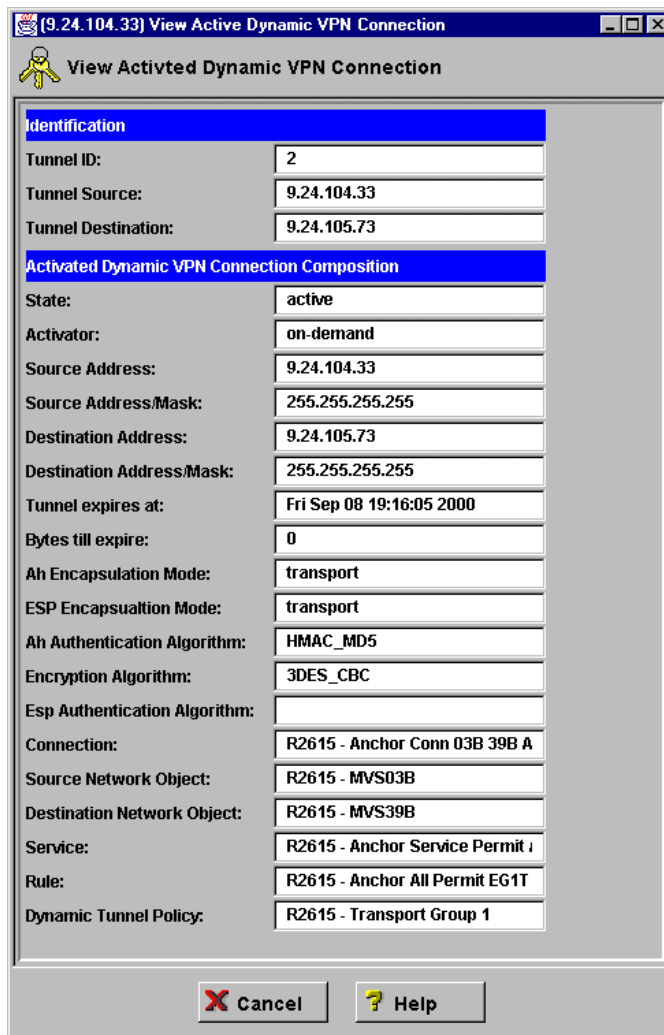


Figure 8-97 z/OS 03B: on-demand tunnel to z/OS RA39 TCIPB detail view

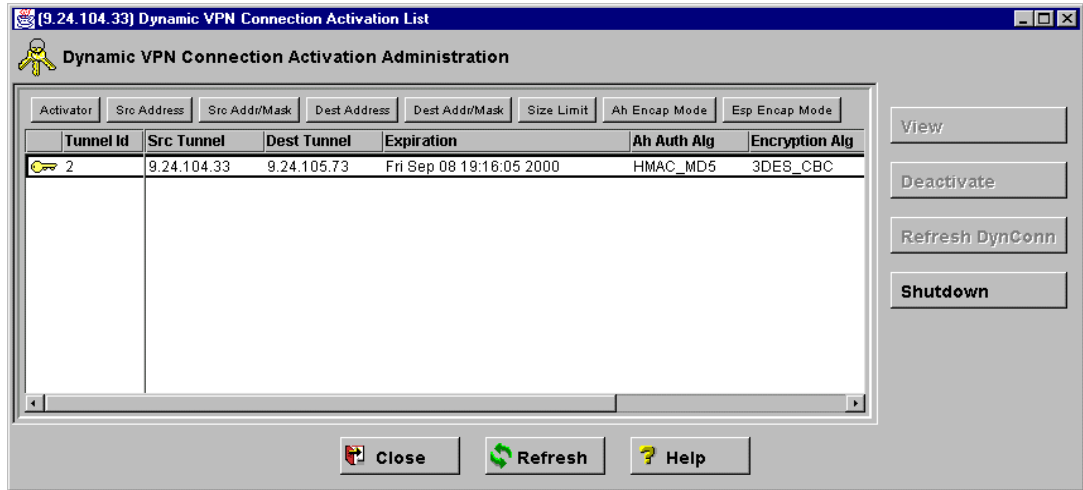


Figure 8-98 z/OS 03B: on-demand tunnel to z/OS RA39 TCPIP list view

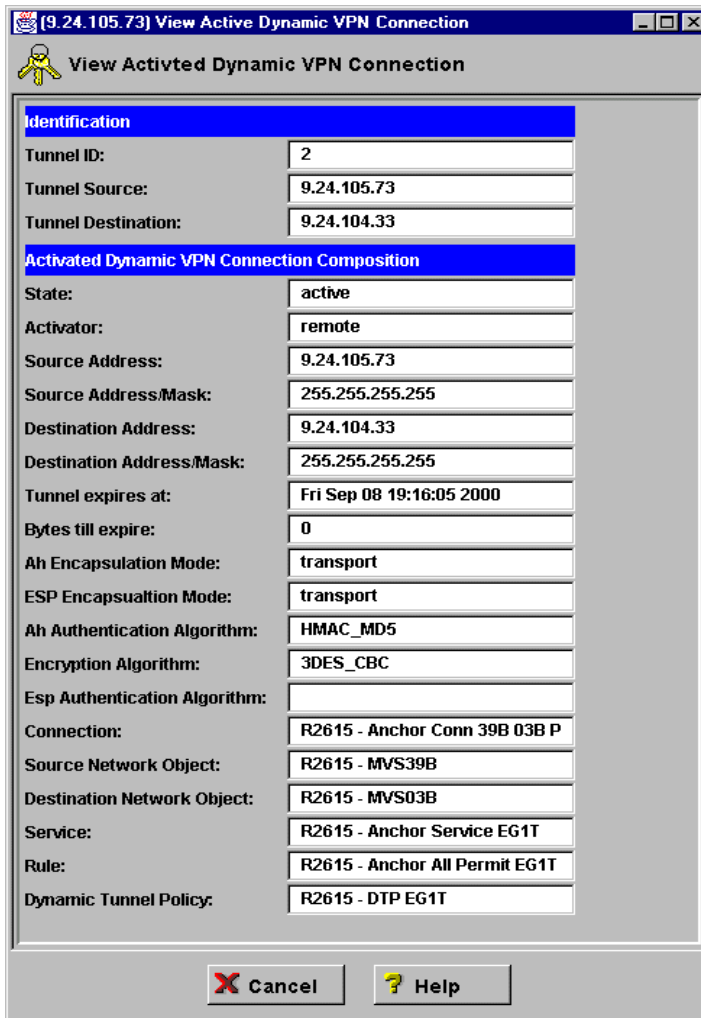


Figure 8-99 z/OS RA39 TCPIP: on-demand tunnel to z/OS 03B detail view

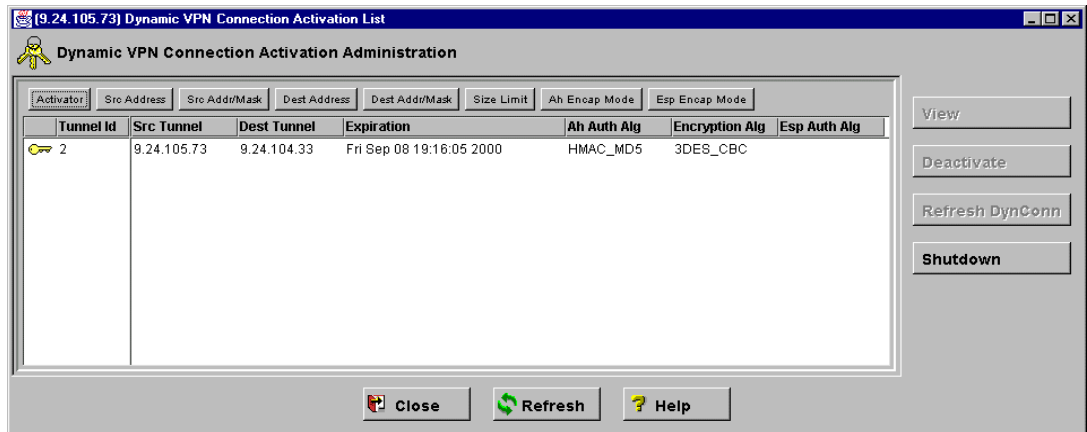


Figure 8-100 z/OS RA39 TCPIPB: on-demand tunnel to z/OS 03B list view

Look at the tunnel characteristics: on the 03B side the activator field contains the on-demand value and on the RA39 TCPIPB side the activator field contains the value remote. That means that the tunnel was created on-demand.

The messages below appear on the ISAKMPD job log. The first one is the 03B and show the creation of the tunnel. These messages are very useful for problem determination.

```
ICA8298i: Attempting to create on demand connection from 9.24.104.33 to 9.24.105.73.
ICA8227i: Dynamic tunnel 000000002 created from 9.24.104.33:255.255.255.255
protocol:ALL port:ALL to 9.24.105.73:255.255.255.255 protocol:ALL port:ALL
encr_alg:3DES_CBC encr_authalg:NONE encr_mode:TRANSPORT authalg:HMAC_MD5
auth_mode:TRANSPORT SA lifetime:3600 SA lifeseize:NONE connection lifetime:86400
PFS:GROUP1.
```

```
ICA8227i: Dynamic tunnel 000000002 created from 9.24.105.73:255.255.255.255
protocol:ALL port:ALL to 9.24.104.33:255.255.255.255 protocol:ALL port:ALL
encr_alg:3DES_CBC encr_authalg:NONE encr_mode:TRANSPORT authalg:HMAC_MD5
auth_mode:TRANSPORT SA lifetime:3600 SA lifeseize:NONE connection lifetime:86400
PFS:GROUP1.
```




Part 4

Application security



Tools for application security

This chapter discusses the ways to secure application traffic. You will find that each of these protocols is used by many applications. For instance, SSL is used by TN3270, FTP, Policy Agent, LDAP, and so on. Each protocol is explained in as much detail as needed to understand the function and references are given for more advanced study.

SSL and TLS use public key cryptography to establish a secret key, which is then used for secret key (or symmetric) cryptography. These protocols require digital certificates for the server, and optionally for the client. For information on the SSL and TLS protocols, see 9.1, “Secure Sockets Layer (SSL)” on page 186.

For a brief overview of the differences between SSL and TLS, see 9.2, “TLS protocol” on page 191.

The Kerberos system is a secret key system that uses symmetric keys, one at the client and another at what is known as a Key Distribution Center (KDC). z/OS applications that can make use of Kerberos are FTP (server and client), UNIX Telnet and UNIX rsh. See 9.3, “Kerberos-based security system” on page 192.

9.1 Secure Sockets Layer (SSL)

The first version of the Secure Sockets Layer protocol was developed by Netscape Communications Corp in 1994 to enable secure Web transactions. Since then, the SSL protocol has been widely deployed to protect traffic for a number of different applications. In 1996 Netscape Communications handed the responsibility for SSL (by now at Version 3.0) over to the Internet Engineering Task Force (IETF), who enhanced the protocol and released it as TLS V1.0. TLS is discussed in 9.2, “TLS protocol” on page 191.

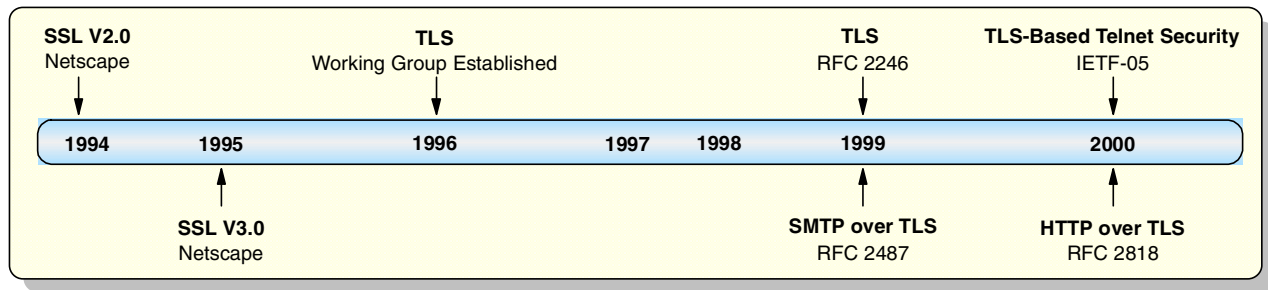


Figure 9-1 Evolution of SSL

SSL-enabled applications on z/OS comprise:

- ▶ IBM HTTP Server for z/OS
- ▶ TN3270 Server
- ▶ z/OS LDAP Server
- ▶ z/OS Firewall Configuration Server
- ▶ CICS Web Interface
- ▶ z/OS UNIX policy agent
- ▶ Digital Certificate Access Server (DCAS) used in the Express Logon Feature

TLS-enabled applications on z/OS comprise FTP server and client.

SSL relies on digital certificates and a hierarchy of trusted authorities, as described in 2.2.4, “Digital certificates” on page 13, to ensure authentication of clients or servers.

9.1.1 SSL protocol description

The SSL protocol defines the partners of a conversation as either a “client” or a “server”. This terminology is used because a client must send certain sets of messages and the server responds with another set. The SSL protocol begins with a “handshake” initiated by the client. During the handshake, the client authenticates the server, the server optionally authenticates the client, and the client and server agree on encryption and authentication algorithms.

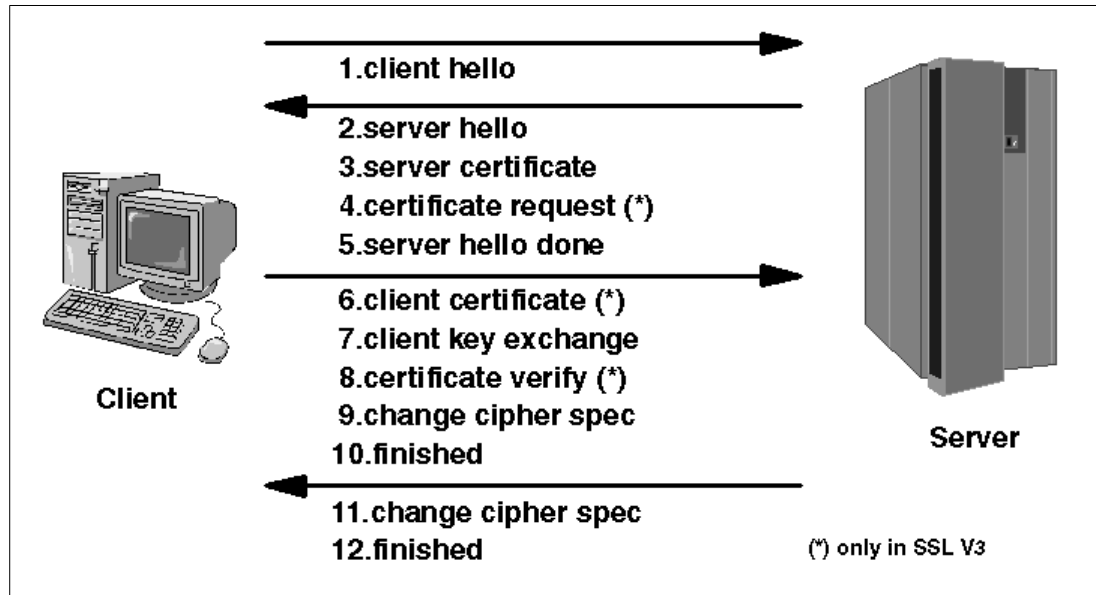


Figure 9-2 Overview of SSL handshake protocol

1. First, the client sends a `client hello` message, which lists the cryptographic capabilities of the client (sorted in client preference order) and contains the SSL/TLS protocol version desired. It also contains a random number used later to generate a secret key by both server and client, a session ID (used for resumed sessions, not discussed here), and a list of cipher suites that the client can support. A cipher suite is an entry indicating an encryption algorithm and a message hashing algorithm.
2. The server responds with a `server hello` message, which contains the cipher suite selected by the server, the session ID, another random number, and the acceptable SSL/TLS protocol version. The client and server must support at least one common cipher suite or the handshake will fail.
3. Following the `server hello` message, the server sends its certificate. This message contains the server's digital certificate and all other certificates up to the "root". The whole chain of certificates is included because the client must match the issuers of the certificates all the way up to the root certificate to find a match with an issuer that it trusts. In a z/OS system server, the certificate is obtained from one of three sources: a keyring database stored in an HFS or MVS data set, which is created with the `gskkyman` utility, or the RACF database using the `DIGTCERT` class.
4. If SSL Version 3 or later (TLS) is used and the server application requires a certificate for client authentication, the server sends a `certificate request` message. In the `certificate request` message, the server sends a list of the types of certificates supported and the distinguished names of acceptable certification authorities.
5. The server then sends a `server hello done` message and waits for a client response. Upon receipt of the `server hello done` message, the client verifies the validity of the server's certificate and checks that the server hello parameters are acceptable.
6. If the server requested a client certificate, the client sends a `certificate` or, if no suitable certificate is available, a `no certificate alert`. This alert is only a warning, but the server application can fail the session if client authentication is mandatory. If a certificate is available, this message contains the client's digital certificate and all other certificates up to the "root". The whole chain of certificates is included because the server must match the issuers of the certificates all the way up to the root certificate to find a match with an issuer that it trusts.

7. The client then sends a `client key exchange` message. This message contains the so-called pre-master secret, a 46-byte random number that is used in the generation of the symmetric encryption keys and the message authentication code (MAC) keys, encrypted with the public key of the server.
8. If the client sent a certificate to the server, the client will now send a `certificate verify` message, which is signed with the client's private key. By verifying the signature of this message, the server can explicitly verify the ownership of the client certificate.

A similar process to verify the server certificate is not necessary. If the server does not have the private key that belongs to the certificate, it cannot decrypt the pre-master secret nor create the correct keys for the symmetric encryption algorithm, and the handshake must fail.
9. Now, the client uses a series of cryptographic operations to convert the premaster secret into a master secret, from which all key material required for encryption and message authentication is derived. Then, the client sends a `change cipher spec` message to make the server switch to the newly negotiated cipher suite.
10. The `finished` message immediately following is the first message encrypted with this cipher method and keys.
11. After the server responds with a `change cipher spec` and a `finished` message of its own, the SSL handshake is completed and encrypted application data can be sent.

The SSL Record Protocol transfers application data using the encryption algorithm and keys agreed upon during the handshake phase. As explained above, symmetric encryption algorithms are used, since they provide much better performance than asymmetric algorithms.

9.1.2 Certificates for SSL

To conduct commercial business on the Internet, you might use a widely known Certificate Authority (CA), such as VeriSign, to get a high assurance server certificate. For a relatively small private network within your own enterprise or group, you can issue your own server certificates, called self-signed certificates, using the z/OS UNIX `gskkyman` utility or the RACF `RACDCERT` command.

In SSL, servers are always authenticated by the client. This means that the client must have access to a CA certificate that can verify the servers certificate.

Client authentication, which is optional, provides additional authentication and access control by checking client certificates at the server. This support prevents a client from obtaining a connection without an installation approved certificate. There are three levels of client authentication:

► Level 1

The authentication is performed by system SSL and ensures that the server's keyring contains a CA certificate that can verify the client certificate. For information about digital certificates, see 2.2.4, "Digital certificates" on page 13.

► Level 2

The authentication provides, in addition to level 1 support, that the client certificate be registered with RACF (or other SAF compliant security product) and mapped to a RACF user ID in the RACF database.

► Level 3

The authentication provides, in addition to level 1 and level 2 support, the capability to restrict access to a server (and port number) based on a profile that can be set up in RACF. The user ID that is associated with the client certificate is tested for access rights to the server and port represented by the RACF profile.

Figure 9-3 shows an example of a z/OS server (TN3270 in this case) using SSL.

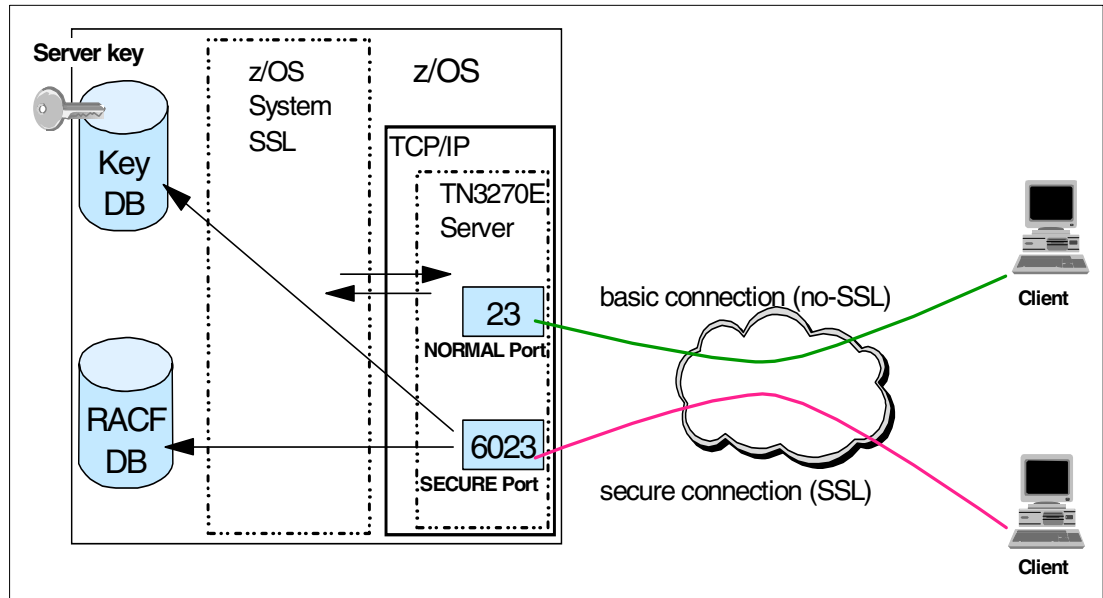


Figure 9-3 SSL protocol

The client must verify the server's certificate based on the certificate of the Certificate Authority (CA) that signed the certificate or based on a self-signed certificate from the server. The server must verify the client's certificate (if client authentication has been configured in the server) using the certificate of the CA that signed the client's certificate. The client and the server then use the negotiated session keys and begin encrypted communications.

A program may require a certificate associated with itself depending on what side of the SSL connection the program is running. This requirement also depends on whether client authentication is requested as part of the SSL handshake. Programs acting as SSL servers (act as the server side of the SSL handshake protocol) must have a certificate to use during the handshake protocol. A program acting as an SSL client requires a certificate in the key database if the SSL server requests client authentication as part of the SSL handshake operation.

If the organization chooses to use a Certificate Authority (within the organization or outside of the organization), then you must generate a certificate request. If only self-signed server certificates are used, you do not have to formulate a certificate request to be sent to an external Certificate Authority (CA) for approval. However, in this case SSL clients do have to import the server's self-signed certificate so that it can be verified during SSL handshake processing.

Additional information about the concepts of cryptography and SSL can be found at the following Web sites:

- <http://home.netscape.com/eng/ss13>
- <http://www.verisign.com/repository/crptintr.html>

Refer to Appendix 10, “Certificate management in z/OS” on page 203 for steps regarding the creation of certificates with gskkyman and RACF.

9.1.3 System SSL

System SSL is a common set of libraries for use by clients and servers in a z/OS system. An Application Programming Interface (API) is provided by System SSL in order to use the SSL code library.

System SSL is part of the System SSL Cryptographic Services Base element of z/OS. z/OS CS IP uses the System SSL APIs to create and manage SSL connections. X.509 certificates are used by both the client and server when securing communications using System SSL.

System SSL supports the following two methods for managing PKI private keys and digital certificates:

- ▶ A z/OS shell-based program called gskkyman. gskkyman creates, fills in, and manages a z/OS HFS file that contains PKI private keys, certificate requests, and certificates. This z/OS HFS file is called a key database and, by convention, has a file extension of *.kdb*.
- ▶ The z/OS SecureWay Security Server (RACF) RACDCERT command. RACDCERT installs and maintains PKI private keys and certificates in RACF. RACF supports multiple PKI private keys and certificates to be managed as a group. These groups are called keyrings. RACF keyrings are the preferred method for managing PKI private keys and certificates for System SSL.

Table 9-1 shows the encryption capabilities of each System SSL fmid. The FMIDs in tgh table refer to z/OS V1R2.

Table 9-1 SSL encryption capabilities

Encryption Type / Key Sizes	Base Security Level FMID HCPT320	Security Level 3 FMID JCPT321
512 bit keys	X	X
1024 bit keys	X	X
1 - SSL V2.0 RC4 US		X
2 - SSL V2.0 RC4 Export	X	X
3 - SSL V2.0 RC2 US		X
4 - SSL V2.0 RC2 Export	X	X
6 - SSL V2.0 DES 56-Bit	X	X
7 - SSL V2.0 Triple DES US		X
01 - SSL V3.0 NULL MD5	X	X
02 - SSL V3.0 NULL SHA	X	X
03 - SSL V3.0 RC4 MD5 Export	X	X
04 - SSL V3.0 RC4 MD5 US		X
05 - SSL V3.0 RC4 SHA US		X
06 - SSL V3.0 RC2 MD5 Export	X	X
09 - SSL V3.0 DES SHA Export	X	X

Encryption Type / Key Sizes	Base Security Level FMID HCPT320	Security Level 3 FMID JCPT321
0A - SSL V3.0 Triple DES SHA US		X

Note: The encryption level used in an SSL connection depends on the client and server encryption level capacity. In the SSL handshake, after server and/or client authentication, both server and client exchange their cipher capabilities and agree on the best cipher algorithm for the session. So, be aware that your TN3270 client must have at least the same level of encryption of your server to have the level of encryption you want.

System SSL supports both the TLS (Transport Layer Security) and SSL (Secure Sockets Layer) protocols.

To implement SSL connections, TCP/IP must have APF authorized access to the System SSL DLLs. The System SSL DLLs are located in hlq.SGSKLOAD by default. System SSL uses the C runtime library (SCEERUN) and the C/C++ IBM Open class library (SCLBDLL), which must also be accessible to TCP/IP. To access these libraries, either add them to the linklist or specify them in the TCP procedure's STEPLIB. If accessed via the linklist, the linklist must be authorized (LNKAUTH=LNKLST specified in the IEASYSxx parmlib member) or the libraries explicitly APF authorized. If accessed via a STEPLIB, the libraries must be APF authorized and DISP=SHR specified.

SSL considerations

As discussed, security functions such as SSL are needed to send sensitive data safely if you connect your system to an insecure network such as the Internet. On the other hand, using such security functions has performance impacts, including utilizing additional CPU cycles and degrading server performance.

To maintain SSL security you have to manage the key carefully, especially when using self-certification, because the whole system environment is affected by the security of the Certificate Authority's key database. On z/OS the key database or keyring file, including the server key pair, may be stored in a file in the HFS if you use the gskkyman utility to manage certificates and keys. In this case, the file may be accessible by users of the z/OS UNIX shell unless you are very careful about setting the UNIX file permission bits on the HFS files, and you do not allow users to enter superuser state. However, RACF is a more secure environment to store certificates and keys, and should be used if possible.

9.2 TLS protocol

SSL 3.0 has outgrown the scope of being a Netscape standard. Continued development of the protocol became the responsibility of the Internet Engineering Task Force in 1996. As a result, SSL 3.0 evolved into the proposed standard for Transport Layer Security, RFC 2246.

TLS is the latest in the continuing evolution of SSL. TLS 1.0 might as readily have been titled SSL 3.1. In fact, when negotiating a TLS handshake, the client and server hello messages will use version specification 3.1 (SSL 3.0 uses version specification 3.0).

Enhancements from SSL V3.0 to TLS V1.0 include:

- ▶ Additions to the number of "alert" messages defined in the protocol
- ▶ Standardized method of calculating message authentication codes (MAC)
- ▶ Simplified CertificateCertify message

- ▶ Simplified Finished message

9.3 Kerberos-based security system

Kerberos is a network authentication protocol that was developed in Project Athena at the Massachusetts Institute of Technology, in cooperation with IBM and Digital Equipment Corporation in the 1980s. DES cryptography is used to provide data privacy, especially for sensitive data such as passwords to log into a server.

Kerberos Version 5 is the latest release and has been implemented in SecureWay Security Server Network Authentication and Privacy Service for z/OS, and chosen by Microsoft Corporation as their preferred authentication technology in Windows 2000.

The Kerberos system is an encryption-based security system that provides mutual authentication between the users and the servers in a network environment. The assumed goals for this system are:

- ▶ Authentication to prevent fraudulent requests and/or responses between users and servers that must be confidential and on groups of at least one user and one server.
- ▶ Authorization can be implemented independently from the authentication by each service that wants to provide its own authorization system. The authorization system can assume that the authentication of a user/client is reliable.
- ▶ Message confidentiality may also be used that provides assurance to a data sender that the message's content is protected from access by entities other than the context's named peer.

Kerberos authentication is based on shared secrets, which are passwords stored on the Kerberos server and client. Those passwords are encrypted with a symmetric cryptographic algorithm, which is DES in this case, and decrypted when needed. This fact implies that a decrypted password is accessed by the Kerberos server, which is not usually required in an authentication system that exploits public key cryptography. Therefore the servers must be placed in locked rooms that are physically secure to prevent an attacker from stealing a password.

For the complete description about the Kerberos Version 5 protocol, refer to *RFC 1510 - The Kerberos Network Authentication Service (V5)*.

9.3.1 Kerberos protocol overview

The Kerberos system consists of three components: a client, a server, and a trusted third party, which is also known as a Key Distribution Center (KDC). KDC interacts with both a client and a server to accept the client's request, authenticate its identity, and issue tickets to it.

The domain served by a single KDC is referred to as a *realm*. A *principal identifier* is used to identify each client and server in a realm. The principal name is uniquely assigned for all clients and servers by the Kerberos administrator. All principals must be known to the KDC.

Although the Kerberos protocol consists of several subprotocols, three exchanges would be of most interest to readers. See Figure 9-4 on page 193. The first phase exchange takes place between a client and the authentication server (AS), in which a client asks the AS that knows secret keys of all clients in the realm to authenticate himself and give it a *ticket granting ticket* (TGT) to be used to get a service ticket for an application server it wants to access.

Upon receiving the TGT, the client sends a request, which contains the TGT, for a service ticket to the ticket-granting server (TGS), and wait until a service ticket is returned. Having the session ticket ready, the client can then communicate with the server that is providing a service he wants to use. Optionally the application server can perform further authentication process against the client.

Note: In most Kerberos implementations, the Authentication Server (AS) and the Ticket Granting Server (TGS) are the same server.

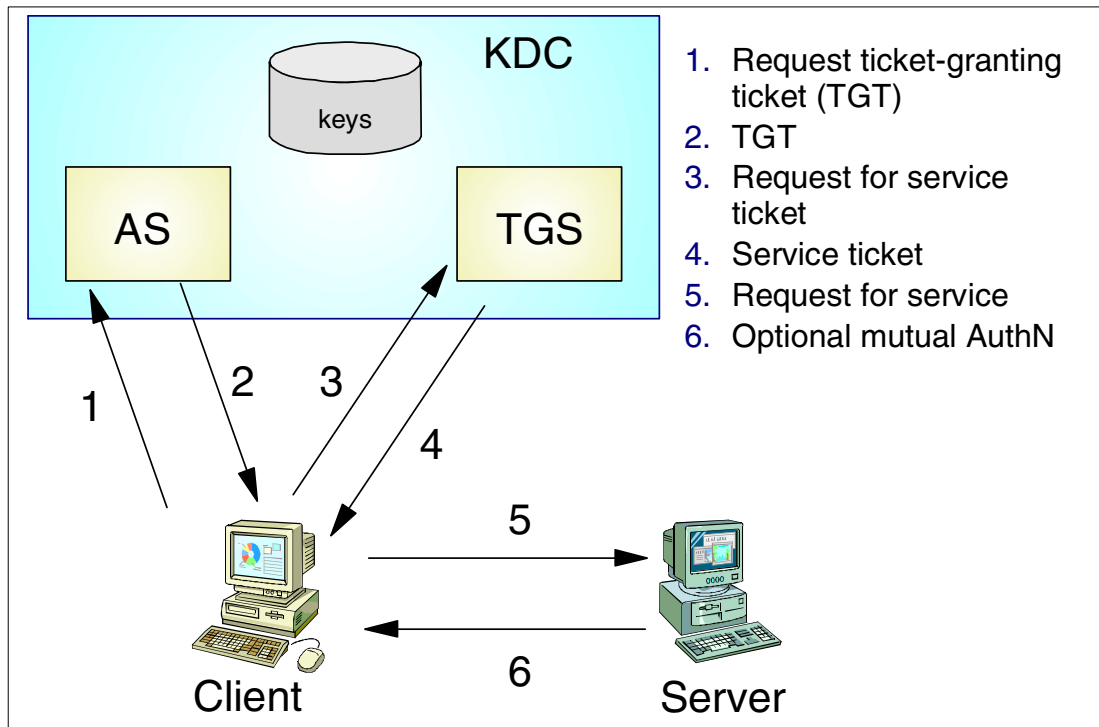


Figure 9-4 Kerberos protocol overview

Message encoding defined in Kerberos Version 5 is described using the Abstract Syntax Notation 1 (ASN.1) syntax in accordance with ISO standards 8824 and 8825.

In the following sections, we discuss the interactions in more detail using the following notations:

- ▶ K_x : X's symmetric encryption key
- ▶ $K_{x,y}$: Encryption key shared by X and Y (for example, a session key)
- ▶ $K_x\{\text{data}\}$: A message that contains data encrypted with X's key

Phase 1: Authentication service (AS) exchange

The authentication service exchange is initiated by a client when it wants to get authentication credentials for an application server but it currently holds no credentials. Two messages are exchanged between the client and the Kerberos authentication server; then credentials for a ticket-granting server (TGS) are given to the client, which is called the *ticket-granting ticket* (TGT) and will subsequently be used to obtain credentials for other services.

This exchange is also used for other services, such as the password-changing service. As noted in Figure 9-5 on page 194, the client's secret key is used exclusively in this phase.

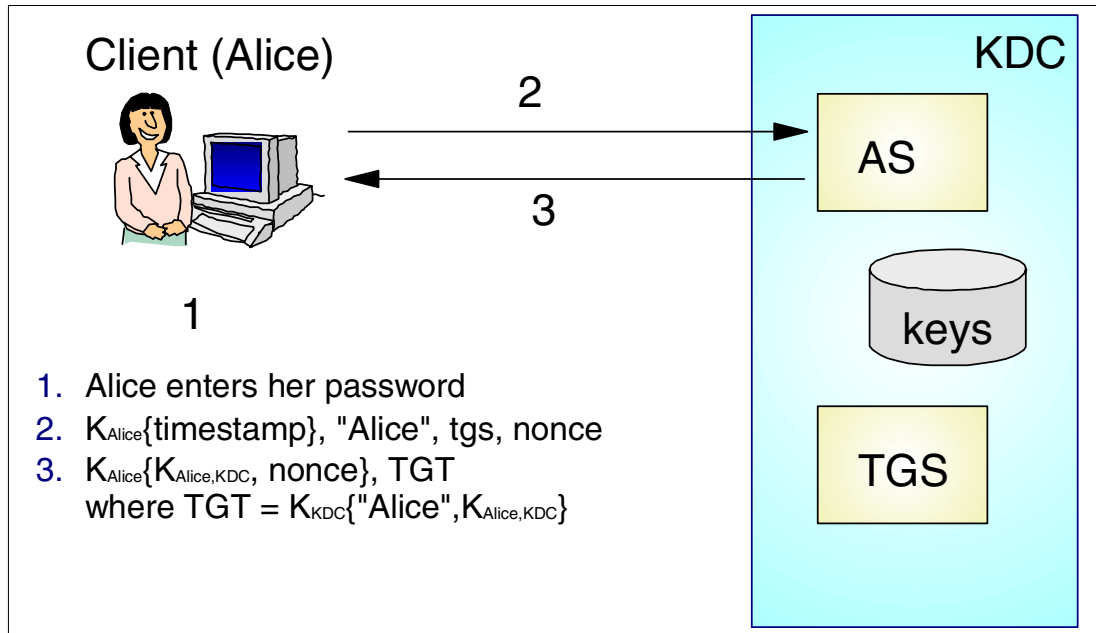


Figure 9-5 Simplified authentication service exchange

When a user logs into a client system and enters her password, a client sends the Kerberos authentication server (AS) a message that includes a user name in plain text ("Alice"), the current time encrypted with her secret key, and the identity of the server for which the client is requesting credentials (TGS in Figure 9-5).

Upon receiving the request from the client, the AS looks up the client name and the service name (the TGS in this case) in the Kerberos database, and then obtains an encryption key of each of them, K_{Alice} and K_{KDC} .

The AS then generates a response back to the client, which contains the TGT and a session key $K_{\text{Alice,KDC}}$, which is used in the subsequent secure communication between the client and KDC. The TGT includes the session key $K_{\text{Alice,KDC}}$, the identities of the server and the client, lifetime, and some other information. The AS then encrypts the ticket using its own key K_{KDC} . This produces a *sealed ticket*. The session key $K_{\text{Alice,KDC}}$ is also encrypted using the client's key K_{Alice} with some other information, such as nonce.

The encrypted current time is also known as the *authenticator*, since the receiver can assure that the sender knows the correct shared secret K_{Alice} , which is the client's *long-term key*, by decrypting it and validating what is inside. Because the AS knows the Alice's secret key, it can evaluate the time decrypted from the received authenticator. As you might have noticed, the clocks on the client system and the KDC must be reasonably synchronized with each other. A network time service may be used for this purpose.

An authenticator is also used to help server detect the message replays.

A *nonce* is information to identify a pair of Kerberos requests and responses. A timestamp or a random number generated by a client may be used.

TGS is the server's identification, which is the Kerberos ticket-granting server (TGS) in this case.

Since K_{Alice} is known exclusively by Alice and KDC, no one but Alice can extract the critical information from the response message, such as the session key $K_{Alice,KDC}$ to be used in the next phase.

When the client receives the AS's response, it decrypts it using its secret key K_{Alice} and checks to see if the nonce matches the specific request. If the nonce matches, the client caches the session key $K_{Alice,KDC}$ for future communications with the TGS.

Phase 2: Ticket-granting service (TGS) exchange

The next phase is used for a client to obtain credentials for services that it wants to use. This exchange is also initiated by the client, and two messages are exchanged between the client and the ticket-granting server (TGS). The protocol and message format used in this exchange is almost identical to those for the AS exchange. The primary difference is that the client's key is never used in this exchange, but the session key obtained from the preceding AS exchange is used.

The request message the client sends to the TGS contains several pieces of information including:

- ▶ Information to authenticate the client, which includes a new authenticator and the TGT obtained from the preceding AS exchange
- ▶ Identity of the service for which the client is requesting credentials
- ▶ Nonce to identify this request

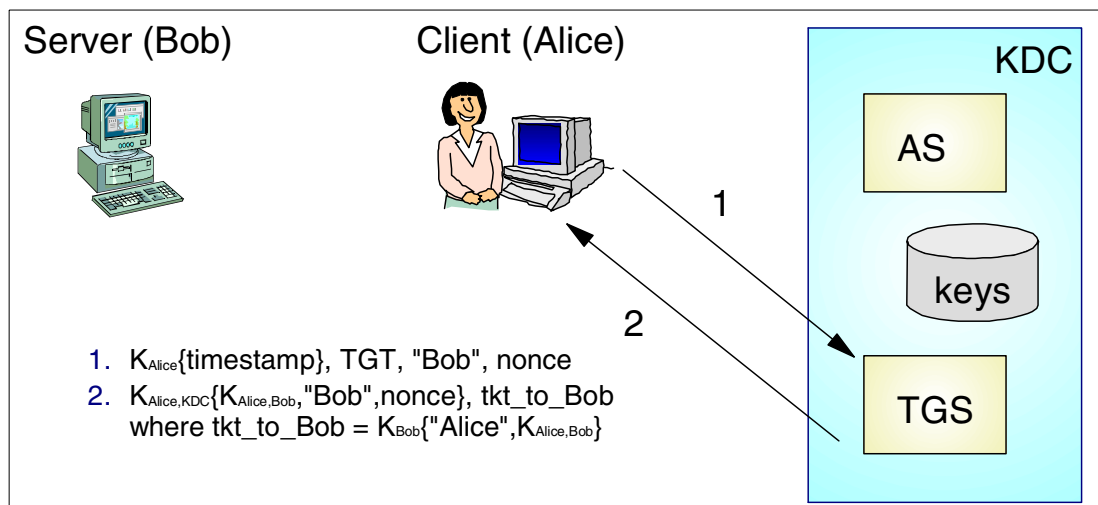


Figure 9-6 Simplified ticket-granting service exchange

When the ticket-granting server (TGS) receives the above message from the client, it first decipheres the sealed ticket using its encryption key K_{KDC} . From the deciphered ticket, the TGS obtains the session-key $K_{Alice,KDC}$. It uses this session key to decipher the authenticator. The validity checks performed by the TGS include:

- ▶ If the client name and its realm in the ticket match the same fields in the authenticator.
- ▶ If the address from which this message is originated is found in the address field in the ticket, which specifies addresses from which the ticket can be used.
- ▶ If the user-supplied checksum in the authenticator matches the contents of the request. This procedure guarantees the integrity of the message.

Finally, it checks the current time in the authenticator to make certain the message is recent. Again, this requires that all the clients and servers maintain their clocks within some prescribed tolerance.

Note: By checking the timestamp in the nanoseconds scale, the replay attacks can be detected.

The TGS now looks up the server name from the message in the Kerberos database, and obtains the encryption key K_{Bob} for the specified service.

The TGS forms a new random session key $K_{Alice,Bob}$ for the benefit of the client (Alice) and the server (Bob), and then creates a new ticket tk_{to_Bob} containing:

- ▶ The session key $K_{Alice,Bob}$
- ▶ Identities of the service and the client
- ▶ Lifetime

Note: The format of the ticket for a particular service is identical to one of the ticket-granting ticket (TGT).

It then assembles and sends a message to the client.

Phase 3: The client/server authentication (CS) exchange

The client/server authentication (CS) exchange is performed by the client and the server to authenticate each other. The client must have obtained credentials for the server using the AS or TGS exchange before the CS exchange is initiated.

After receiving the TSG exchange response from the TGS, the client deciphers it using the TGS session key $K_{Alice,KDC}$ that is exclusively known by the client and the TGS. From this message it extracts a new session key $K_{Alice,Bob}$ that is shared with the server (Bob) and the client (Alice). The sealed ticket included in the response from the TGS cannot be deciphered by the client because it is enciphered using the server's secret key K_{Bob} .

Then the client builds an authenticator and seals it using the new session key $K_{Alice,Bob}$. At last, it sends a message containing the sealed ticket and the authenticator to the server (Bob) to request its service.

When the server (Bob) receives this message, it first deciphers the sealed ticket using its encryption key K_{Bob} , which is kept in secret between Bob and the KDC. It then uses the new session key $K_{Alice,Bob}$ contained in the ticket to validate the authenticator in the same way as the TGS does in the TGS exchange.

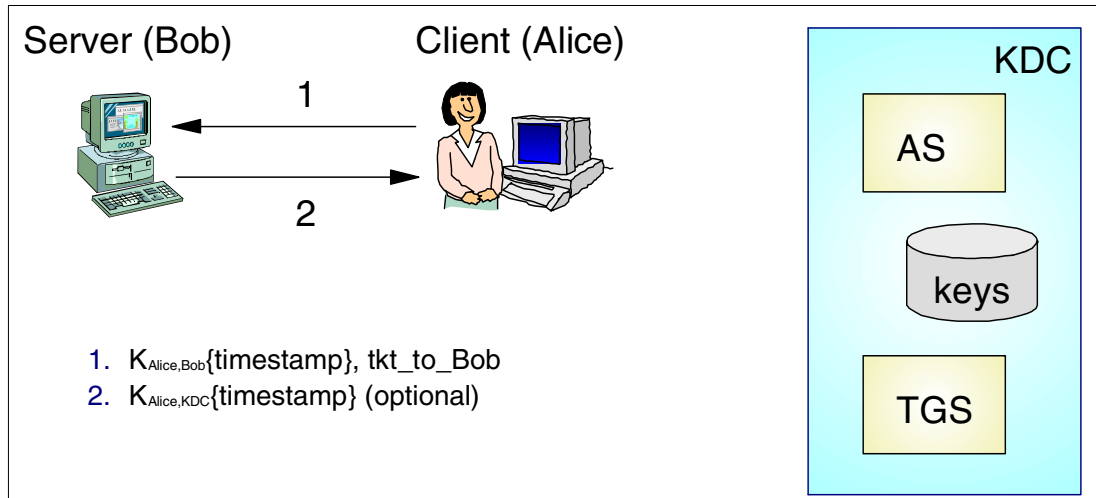


Figure 9-7 Simplified client/server authentication exchange

Once the server has authenticated a client, an option exists for the client to validate the server (this procedure is called *mutual authentication*). This prevents an intruder from impersonating the server.

If mutual authentication is required by the client, the server has to send a response message back to the client. The message has to contain the same timestamp value as one in the client's request message. This message is enciphered using the session key $K_{\text{Alice,Bob}}$ that was passed from the client to the server.

If the response is returned, the client decrypts it using the session key $K_{\text{Alice,Bob}}$ and verifies that the timestamp value matches one in the authenticator that was sent by the client in the preceding CS exchange. If it matches, then the client is assured that the server is genuine.

Once the CS exchange has completed successfully, an encryption key is shared by the client and server and can be used for the on-going application protocol to provide the data confidentiality.

9.3.2 Inter-realm operation

The Kerberos protocol is designed to operate across organizational boundaries. Each organization wishing to run a Kerberos server establishes its own realm. The name of the realm in which a client is registered is part of the client's name and can be used by the application server to decide whether to honor a request.

By establishing inter-realm keys, the administrators of two realms can allow a client authenticated in one realm to use its credentials in the other realm. The exchange of inter-realm keys registers the ticket-granting service of each realm as a principal in the other realm. A client is then able to obtain a ticket-granting ticket for the remote realm's ticket-granting service from its local ticket-granting service. Tickets issued to a service in the remote realm indicate that the client was authenticated from another realm.

This method can be repeated to authenticate throughout an organization across multiple realms. To build a valid authentication path to a distant realm, the local realm must share an inter-realm key with the target realm or with an intermediate realm that communicates with either the target realm or with another intermediate realm.

Realms are typically organized hierarchically. Each realm shares a key with its parent and a different key with each child. If an inter-realm key is not directly shared by two realms, the hierarchical organization allows an authentication path to be easily constructed. If a hierarchical organization is not used, it may be necessary to consult some database in order to construct an authentication path between realms.

Although realms are typically hierarchical, intermediate realms may be bypassed to achieve cross-realm authentication through alternate authentication paths. It is important for the end-service to know which realms were transited when deciding how much faith to place in the authentication process. To facilitate this decision, a field in each ticket contains the names of the realms that were involved in authenticating the client.

9.3.3 Some assumptions

The following limitations are applied to the Kerberized security environment:

- ▶ *Denial-of-service (DoS)* attacks are not addressed by Kerberos. There are places in these protocols where an intruder can prevent an application from participating in the proper authentication steps. Detection and solution of such attacks (some of which can appear to be “usual” failure modes for the system) is usually best left to human administrators and users.

Note: TRMD may be used to protect against such attacks as resource hogging.

- ▶ The secret key must be kept in secret by each principal (each client and server). If an attacker steals a principal’s key, it can then masquerade as that principal or impersonate any server of the legitimate principal.
- ▶ Kerberos does not address *password-guessing* attacks. If a poor password is chosen, an attacker may be able to mount an offline dictionary attack by repeatedly attempting to decrypt messages that are encrypted with a key derived from the user’s password.
- ▶ Kerberos assumes a loosely synchronized clock in the whole system. Workstations may be required to have a synchronization tool such as the time server provided.
- ▶ Principal identifiers should not be reused on a short-term basis. Access control lists (ACLs) may be used to grant permissions to particular principals.

9.3.4 Kerberos implementation in z/OS

The Kerberos Version 5 server has been introduced in OS/390 V2R10 and implemented in SecureWay Security Server Network Authentication Service for z/OS. Kerberos provides strong authentication and encryption for the following applications:

- ▶ The UNIX Telnet server - authentication support provided by the Kerberos 5 protocol
- ▶ The UNIX remote shell execution (rsh) server - authentication support provided by the Kerberos 5 protocol and the GSSAPI protocol
- ▶ The FTP client and FTP server - authentication support provided by the GSSAPI protocol

Note: The Kerberos server shipped with CS for z/OS IP supports Kerberos Version 4 and is discontinued in z/OS V1R2.

Restriction: The zSeries KDC is incompatible with Windows 2000 Kerberos applications. Windows 2000 applications must use the Windows KDC. To support Windows 2000 applications, a cross-realm connection between the zSeries KDC and the Windows KDC is required.

The following is a brief overview of how Kerberos is set up in z/OS.

RACF support for Kerberos

The Kerberos realm and its trust relationships with other realms is defined using the general resource class REALM. To define the local realm, you set up a REALM class profile named KERBDFLT. Figure 9-8 shows a local realm ZOS12.RAL.IBM.COM being defined with a minimum ticket lifetime of 30 seconds, a default ticket lifetime of 10 hours, a maximum ticket lifetime of 24 hours, and a password of NEW1PW. All of the ticket lifetimes are specified in seconds. The administrator then lists the new REALM profile with the RACF RLIST command.

```

RDEFINE REALM KERBDFLT KERB(KERBNAME(ZOS12.RAL.IBM.COM) -
      PASSWORD(kerberos) MINTKTLFE(15) DEFTKTLFE(36000) -
      MAXTKTLFE(86400))

      RLIST REALM KERBDFLT KERB NORACF
CLASS      NAME
-----
REALM      KERBDFLT

KERB INFORMATION
-----
KERBNAME=  ZOS12.RAL.IBM.COM
MINTKTLFE= 0000000015
MAXTKTLFE= 0000086400
DEFTKTLFE= 0000036000
KEY VERSION= 001
KEY ENCRYPTION TYPE= DES DES3 DESD
***
```

Figure 9-8 Setting up the local Kerberos realm using the RACF REALM class

A Kerberos principal is defined in the KERB segment of a user profile (in the same way that the UNIX information for a user is stored in the OMVS segment). You can use the RACF ADDUSER (for new users) or ALTUSER (for existing users) commands to add the KERB segment for a user ID. In the KERB segment, the KERBNAME parameter identifies the local principal name. Local principal names may contain imbedded blanks and lowercase characters, and must be unique. For instance:

```
a1u focas password(kerbpass) noexpired kerb(kerbname(FOCAS))
```

associates the RACF user focas with Kerberos principal name FOCAS.

When you add a KERB segment to a user profile, RACF automatically sets up a profile in the KERBLINK class named with the KERBNAME parameter from the user's KERB profile. This enables RACF to have a mapping to a RACF user ID from a Kerberos principal name (which may or may not be the same). When you use ALTER NOKERB to remove a KERB segment from a user, or you use DELUSER to delete a user with a KERB segment, the KERBLINK profile is automatically deleted.

Note: Do not execute the DELUSER command, or an ALTUSER command with the NOKERB option, for a user profile that contains a KERB segment from RACF systems that do not support the KERBLINK class. These systems do not automatically manage KERBLINK profiles. You will inadvertently leave residual mapping profiles in the KERBLINK class. For information about recovery procedures, see *z/OS SecureWay Security Server RACF System Programmer's Guide*, SA22-7681.

Basic steps to follow to configure RACF to support Kerberos are:

- ▶ Customizing the local environment:
 - Defining your local RRSF (RACF remote sharing facility) node.
 - Defining your local realm.
 - Defining local principals.
- ▶ Defining your foreign environment:
 - Defining foreign realms.
 - Mapping RACF user IDs for foreign principals.

The z/OS Kerberos KDC

The Kerberos KDC is implemented by started task SKRBKDC as shown in Figure 9-9. The RACF user ID associated with the started task must have a UNIX UID of UID(0) (a superuser).

```

//*****
//*
//* Procedure for starting the Kerberos Security Server
//*
//*****
//SKRBKDC PROC REGSIZE=256M,OUTCLASS='S'
//*-----
//GO EXEC PGM=EUVFSKDC,REGION=&REGSIZE,TIME=1440,
// PARM=('ENVAR("LANG=en_US.IBM-1047"),TERM(DUMP) X
// / 1>DD:STDOUT 2>DD:STDERR')
//STDOUT DD SYSOUT=&OUTCLASS,DCB=LRECL=250,
// FREE=END,SPIN=UNALLOC
//STDERR DD SYSOUT=&OUTCLASS,DCB=LRECL=250,
// FREE=END,SPIN=UNALLOC
//SYSOUT DD SYSOUT=&OUTCLASS,
// FREE=END,SPIN=UNALLOC
//CEEDUMP DD SYSOUT=&OUTCLASS,
// FREE=END,SPIN=UNALLOC

```

Figure 9-9 Started task for Kerberos server

The SKRBKDC started task reads the Kerberos server configuration file from the SKRBKDC RACF user's OMVS home directory. The Kerberos configuration file specifies which IP host and port the KDC server should be started on for the local realm as well as the IP host and port numbers for KDCs in other realms.


```

;-----
; Sample Kerberos configuration file
;-----

[libdefaults]

default_realm = ZOS12.RAL.IBM.COM 1
kdc_default_options = 0x00000010
use_dns_lookup = 0

; Default encryption types if DES3 is not supported
default_tkt_enctypes = des-cbc-crc
default_tgs_enctypes = des-cbc-crc
; Default encryption types if DES3 is supported
;default_tkt_enctypes = des3-cbc-sha1,des-cbc-crc
;default_tgs_enctypes = des3-cbc-sha1,des-cbc-crc

[realms]

ZOS12.RAL.IBM.COM = {
2 KDC = WTSC63C.ZOS12.RAL.IBM.COM:88
3 KPASSWD_SERVER = WTSC63C.ZOS12.RAL.IBM.COM:464
}

[domain_realm]

.ZOS12.RAL.IBM.COM = ZOS12.RAL.IBM.COM

```

Figure 9-10 Sample Kerberos configuration file

Figure 9-10 shows an example of a Kerberos server configuration file.

- 1** The `default_realm` statement specifies the realm name that is used when a principal wants to start communicating with another principal, and does not specifically state the realm. This should be the DNS root of your system that the KDC will run on.
- 2** The KDC statement for a realm specifies the host name and port number of the KDC server for that realm.
- 3** The `KPASSWD_SERVER` statement for a realm specifies the host name and port number of the Password Change server for that realm.

It should be noted that in the above example, the local realm is `ZOS12.RAL.IBM.COM` (as set up in Figure 9-8 on page 199), and that the host name on the local realm points to where the KDC will be opening a socket. These sockets must be reserved for job name `OMVS` in the TCPIP stack that the server will be running on.

Verifying correct KDC startup

After the `SKRKBKDC` started task has successfully started, check to ensure that the KDC server is listening on the correct sockets in the TCP/IP stack that you have targeted using the `onetstat -s -p stackname OMVS` command.

In your TSO logon proc, ensure the kerberos REXX data set `EUVF.SEUVFEXC` is in the `SYSEXEC` concatenation. In `OMVS`, ensure the `PATH` variable has subdirectory `/usr/lpp/skrb/bin` before any other bin library for the user.

```
FOCAS @ SC63:/pfocas>kinit FOCAS
EUVF06017R Enter password:

FOCAS @ SC63:/pfocas>klist
Ticket cache: FILE:/var/skrb/creds/krbcred_cf635eb0
Default principal: FOCAS@ZOS12.RAL.IBM.COM

Server: krbtgt/ZOS12.RAL.IBM.COM@ZOS12.RAL.IBM.COM
Valid 2002/05/30-10:23:39 to 2002/05/30-20:23:39
FOCAS @ SC63:/pfocas>
```

Figure 9-11 Kerberos installation verification procedure

To get an initial ticket from Kerberos, enter the **kinit** command. The first parameter is the principal name. In Figure 9-11 the principal name that we are getting a ticket for is “FOCAS” (note the case is exactly as that entered on the RACF command used to add the KERB segment for the user).

Note: When using the **kinit** command, the password that you enter must be uppercase. This is because when you add it with the RACF ALTUSER command, RACF translates the password to uppercase.

The password was then entered (it must be in uppercase) and the ticket was received from the KDC. The **klist** command shows a list of “credentials” that the current user has. In this case, there is only one, for the default realm ZOS12.RAL.IBM.COM.

The user is now ready to log onto a Kerberized server, such as otelnetd or FTP.

For information on how to Kerberize a server application, see that server’s documentation.

For further information about the Kerberos server in z/OS, refer to *z/OS V1R2.0 SecureWay Security Server Network Authentication Server Administration*, SC24-5926.



Certificate management in z/OS

In the z/OS environment, digital certificates are used by SSL/TLS to authenticate and encrypt the protocol handshaking messages. An SSL/TLS server must send its certificate to the client, and a server can optionally request the client for a certificate. For the purposes of this chapter, SSL and TLS are equivalent unless stated otherwise.

There are two ways for you to obtain a certificate. One is to request a Certificate Authority (CA) to create your certificate. If you are requesting a certificate for a server, and you plan to make your server available to the public or your business partners, you should get your certificate from a trusted CA such as VeriSign, Inc. or any other Certificate Authority whose root certificate is contained in the key database of the clients who use your server.

The second way for you to obtain a certificate is to generate one yourself. This type of certificate is called a *self-signed* certificate because the issuer of the certificate is the same as the subject of the certificate. This type of certificate might be useful for testing purposes or for securing SSL connections within your intranet.

To validate a certificate, the receiver checks its key database for a trusted CA certificate that has the same distinguished name as that of the received certificate's certifier. Thus the CA certificates must be located in the client's and server's local database (or *keyring*) and marked as trusted. See Figure 10-1 on page 204.

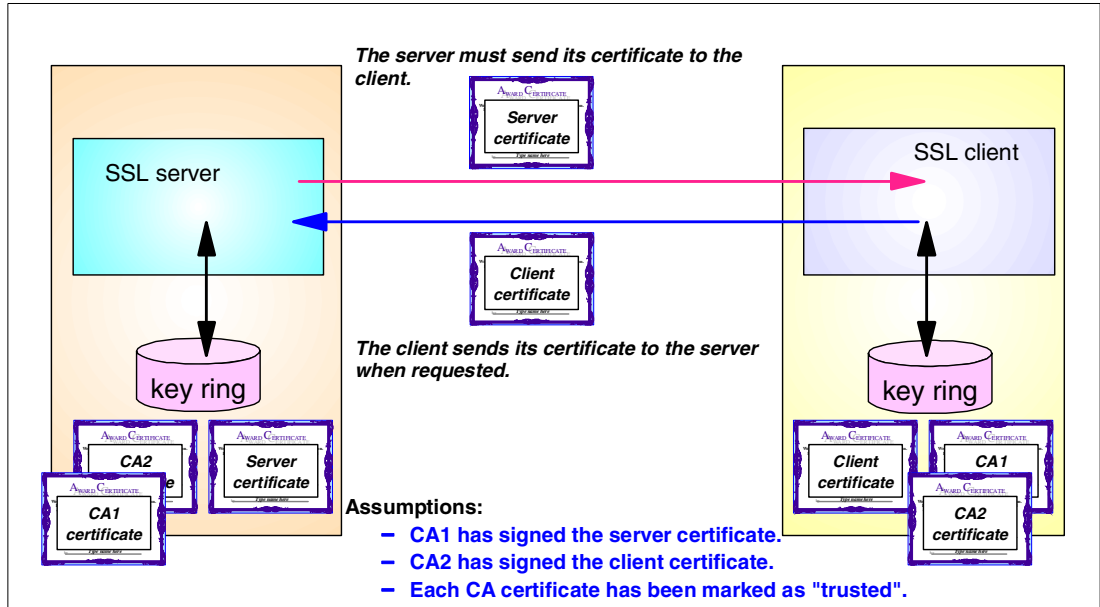


Figure 10-1 SSL certificate management: CA-signed certificates

If you choose to use self-signed certificate instead of CA-signed, the CA certificate that should be trusted is identical to the server/client certificate itself. If the server itself has signed its certificate and client authentication is not required, the server certificate must be exported and stored in the client's local database as a trusted CA certificate, because the server certificate *is* the issuer's certificate for itself.

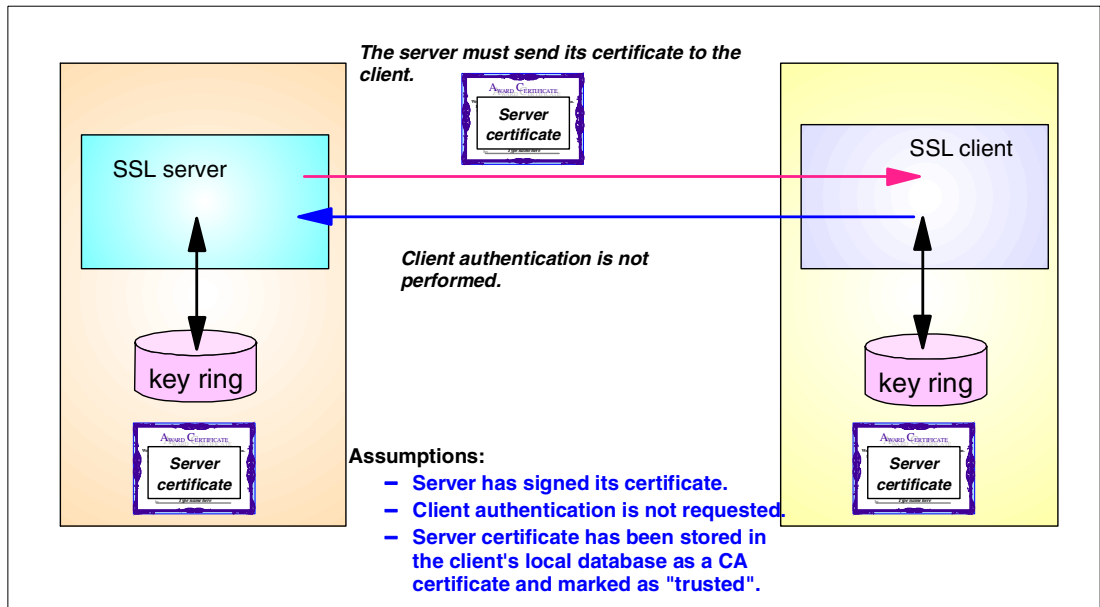


Figure 10-2 SSL certificate management: self-signed certificate without client authentication

10.1 Digital certificates in z/OS

Most SSL-enabled applications in z/OS are making use of the System SSL toolkit as described in “System SSL” on page 190. For certificate storage and management, two command utilities exist:

- ▶ The RACF command RACDCERT, which creates and maintains certificates and keyrings that are stored in the RACF database. This command can also be used to create self-signed certificates and certificate requests for other CAs.
- ▶ The gskkyman utility, which creates and maintains a key database as a file in the HFS. It can also create self-signed certificates and certificate requests for other CAs.

Using RACF keyrings is the preferred method because it provides better security for the certificates and their private keys. With RACF keyrings, stash files containing key database passwords are not used and access to keyrings and certificates is controlled by RACF. In this chapter, we show both methods of creating and managing certificates.

For detailed information regarding the creation and maintenance of digital certificates in z/OS, see Chapter 9, “Certificate/Key Management”, in *z/OS V1R2 System SSL Programming*, SC24-5901. For a reference on the RACDCERT command, see *z/OS V1R2.0 SecureWay Security Server RACF Command Language Reference*, SA22-7687.

Table 10-1 summarizes all applications that make use of the certificate management tools in z/OS V1R2.

Table 10-1 Applications that use digital certificates in z/OS V1R2

		RACDCERT	gskkyman
SSL	TN3270 server	X	X
	HTTP Server	X	X
	Pagent Client		X
	LDAP server	X	X
	Policy agent		X
	DCAS server	X	X
	Firewall configuration client	X	X
TLS	FTP Server	X	X
IKE	IKE server	X	

10.2 Digital certificate field formats

When you create a digital certificate whether using **gskkyman** or **RACDCERT**, certain fields are required and others are optional. We cover the most important fields here.

- ▶ **Certificate Version Number.** This is always “3”. **gskkyman** will ask for the number, while **RACDCERT** sets it automatically.
- ▶ **Distinguished Name.** The issuer of a certificate and the subject of a certificate are both represented by a “distinguished name”. This name takes the form of a hierarchy, although different certificate issuers treat the format differently. For a self-signed certificate, the

Issuer's Distinguished Name will be copied from the Subject's Distinguished Name. A Distinguished Name contains the following subfields (with RACDCERT parameter names in parentheses):

- **Common Name.** (CN). For a server certificate, this field normally contains the server's DNS name. For a client certificate, this will identify the individual or computer.
 - **Organization-name.** (O). Company name or similar.
 - **Title.** (T). Salutation for an individual.
 - **Organizational-unit.** (OU). Used for classification within the "Organization-name", above.
 - **Locality.** (L). City or town.
 - **State-or-province.** (SP).
 - **Country.** (C). Two character ISO code for country.
 - The only compulsory subfields of the Distinguished Name are the Common Name, the Organization Name and the Country.
- **Period of validity.** The gskkyman utility asks for the number of days from today that the certificate is valid for, while RACDCERT sets the lower and upper dates with the NOTBEFORE(DATE(yyyy-mm-dd)) and the NOTAFTER(DATE(yyyy-mm-dd)) parameters.

The Label field is also needed. This field is not part of the X509 specification but it is used to organize certificates in the key database. You can use the label to list, alter, and delete individual certificates. A label must be unique except for storage within RACF, where labels can be duplicated as long as they are associated with different RACF user IDs (with the ID(user..) parameter).

Figure 10-3 shows a batch job used to create a self-signed digital certificate using RACDCERT.

```

//CERTAUTH EXEC PGM=IKJEFT01,DYNAMNBR=30,REGION=4096K
/**
/** Add the top-level self-signed certificate for the certificate
/** authority (ourselves)
/**
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *
RACDCERT CERTAUTH GENCERT -
SUBJECTSDN( O('I.B.M Corporation') -
CN('server.raleigh.ibm.com') -
C('US')) -
NOTAFTER( DATE(2002-08-22) ) -
SIZE(512) -
WITHLABEL('Label for RACDCERT cert')
/**

```

Figure 10-3 Setting up a test self-signed certificate, for a gskkyman comparison

1 The SUBJECTSDN parameter encloses all the Distinguished Name subfields for the subject.

Figure 10-4 on page 207 shows the same certificate being created (with a different label) using gskkyman in order to show how the certificate fields are specified in each utility. The required fields of Common Name, Organization and Country were specified, the key size was set to 512 bits, and a 100-day period of validity was set.

```

Current key database is /example.kdb

  1 - List/Manage keys and certificates
  2 - List/Manage request keys
  3 - Create new key pair and certificate request
  4 - Receive a certificate issued for your request
  5 - Create a self-signed certificate
  6 - Store a CA certificate
  7 - Show the default key
  8 - Import keys
  9 - Export keys
 10 - List all trusted CAs
 11 - Store encrypted database password

  0 - Exit program

Enter option number (or press ENTER to return to the parent menu): 5
Enter version number of the certificate to be created (1, 2, or 3) [3]: 3
Enter a label for this key.....> Label for gskkyman cert
Select desired key size from the following options (512):
  1: 512
  2: 1024
Enter the number corresponding to the key size you want: 1
Enter certificate subject name fields in the following.
  Common Name (required).....> server.raleigh.ibm.com
  Organization (required).....> I.B.M Corporation
  Organization Unit (optional).....>
  City/Locality (optional).....>
  State/Province (optional).....>
  Country Name (required 2 characters)..> US
Enter number of valid days for the certificate [365]: 100
Do you want to set the key as the default in your key database? (1 = yes, 0 = no
) [1]: 0
Do you want to save the certificate to a file? (1 = yes, 0 = no) [1]: 0

Please wait while self-signed certificate is created...

```

Figure 10-4 Example of setting up a certificate in gskkyman

10.3 RACF RACDCERT command use

RACF can be used to create, register, store, and administer digital certificates and the private keys associated with the certificates. RACF can also be used to create and manage keyrings of stored digital certificates. Certificates are stored in the RACF database, while private keys may be stored in the ICSF Public Key Data Set (PKDS), encrypted under a 168-bit Triple-DES key.

RACF distinguishes three types of digital certificates:

- ▶ **Certificate Authority certificates:** these certificates are associated with Certificate Authorities (CAs) and are used to verify signatures in other certificates.
- ▶ **Site certificates:** these certificates are associated with servers or network entities in other locations than the local system.
- ▶ **User certificates:** these certificates are associated with a RACF user ID and are used to authenticate a user's identity.

A user certificate or a certificate that has been connected to a keyring with USAGE(PERSONAL) is the only type of certificate whose private key can be used to create signatures. Therefore, all server certificates for local servers need to be user certificates or they need to be connected to an appropriate keyring with USAGE(PERSONAL).

The RACF ISPF panels can be used to maintain the digital certificates if you do not choose to use the TSO RACDCERT command. Our examples show the TSO commands as they can be submitted in a batch job.

10.4 RACF keyrings

A RACF keyring is a way to logically group together a number of certificates. Certificates can be “connected” to one or more keyrings.

Each keyring is associated with only one user, but the certificates that are connected to that keyring may or may not be the keyring-owner’s certificate.

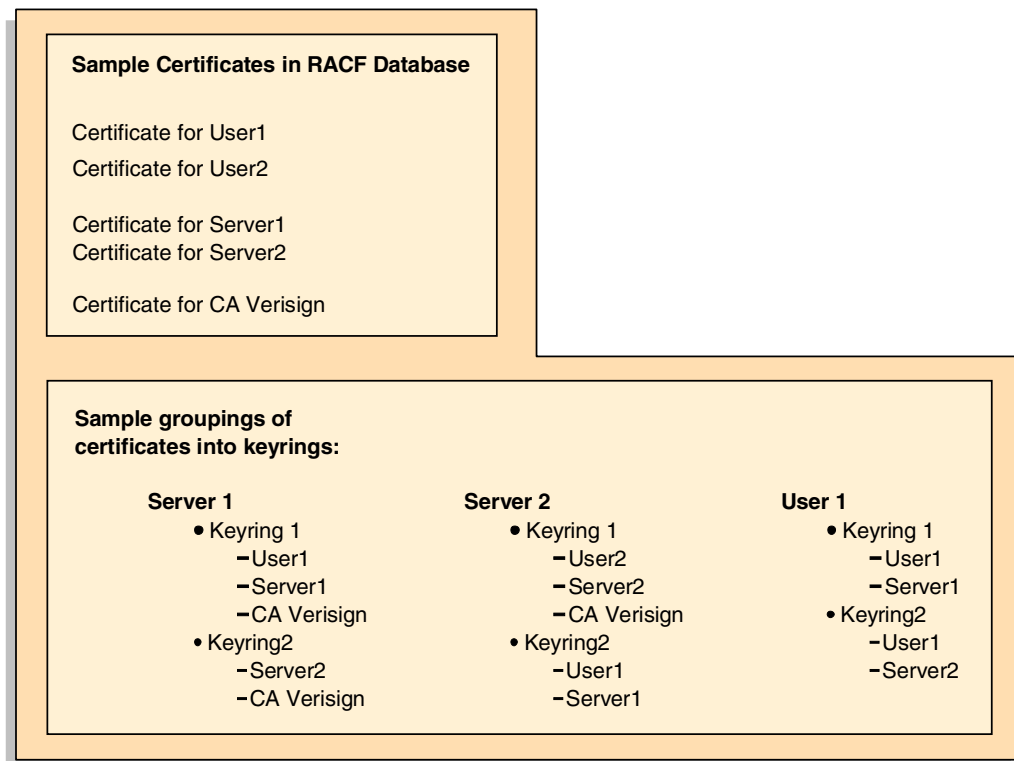


Figure 10-5 Example showing how keyrings contain pointers to certificates

Figure 10-5 shows the logical relationship between RACF keyrings and digital certificates stored in the RACF database. There can be more than one keyring in the database with the same name, but each must be assigned to a different user ID.

Typically, a z/OS server that uses digital certificates will have a configuration parameter where the RACF keyring name is specified. TN3270 and FTP are examples of servers that use keyrings. During SSL/TLS handshaking, the server sends its certificate to the client. The server will get its certificate from the RACF keyring specified in the server’s configuration file and that is associated with the server’s RACF user ID. A server also looks at the certificates in its keyring for a CA certificate with which to validate the client certificate, if client authentication is configured.

A z/OS client that uses digital certificates (such as FTP) will have a configuration parameter where the RACF keyring name is specified. During SSL/TLS handshaking, the server sends its certificate to the client, and the client looks at the certificates in its keyring for a CA certificate with which to validate the server certificate. If the server requests the client certificate, the client will get its certificate from the RACF keyring specified in the client's configuration file and that is associated with the client's RACF user ID. Note that a client certificate must be in TRUSTED status in the RACF database.

10.4.1 RACDCERT command security

Authority to the IRR.DIGTCERT.function resource in the FACILITY class allows a user to issue the RACDCERT command. To issue the RACDCERT command, users must have one of the following RACF authorities:

- ▶ The SPECIAL attribute
- ▶ Sufficient authority to resource IRR.DIGTCERT.function in the FACILITY class
- ▶ READ access to IRR.DIGTCERT.function to issue the RACDCERT command for themselves
- ▶ UPDATE access to IRR.DIGTCERT.function to issue the RACDCERT command for others
- ▶ CONTROL access to IRR.DIGTCERT.function to issue the RACDCERT command for SITE and CERTAUTH certificates (this authority also has other uses)

Important: Any z/OS-based client or server that uses a RACF keyring issues an internal RACDCERT LIST and RACDCERT LISTRING command. The RACF user ID associated with the server must therefore be granted READ access to the RACF profiles controlling these commands, which are IRR.DIGTCERT.LIST and IRR.DIGTCERT.LISTRING. For a list of servers that use RACF keyrings, see Table 10-1.

```
PERMIT IRR.DIGTCERT.LIST CLASS(FACILITY) ID(TCPIPA) ACCESS(READ)
PERMIT IRR.DIGTCERT.LISTRING CLASS(FACILITY) ID(TCPIPA) ACCESS(READ)
SETR RACLIST(FACILITY) REFRESH
```

Figure 10-6 RACF commands to the TCPIP user ID

Figure 10-6 shows the RACF commands needed to permit a user (TCPIPA in this case) to issue the RACDCERT LIST and RACDCERT LISTRING commands.

For more information see *z/OS V1R3.0 Security Server RACF Security Administrator's Guide*, SA22-7683.

10.4.2 RACDCERT command format

RACDCERT [ID(user) | SITE | CERTAUTH] command-options

The RACDCERT command can be directed to a RACF user ID's digital certificates or keyrings by the ID(user) parameter, to a Certificate Authority's resources by the CERTAUTH parameter and to a site's resources by the SITE parameter. If no ID, SITE or CERTAUTH parameter is included, the command issuer's ID is used.

For instance, the command **racdcert certauth list** will list all Certificate Authority certificates in the RACF database, while **racdcert list** shows all of your (the command issuer's) certificates.

There is also a "multiid" parameter for mapping functions. This and other parameters are explained fully in *z/OS V1R3.0 Security Server RACF Security Administrator's Guide*, SA22-7683.

10.5 gskkyman command use

The **gskkyman** UNIX command is used to create and maintain digital certificate key databases in an HFS. This is an alternative to storing digital certificates in the RACF database. Note that if you are using SSL/TLS client authentication to map a digital certificate to a RACF user ID, then you must use the RACF RACDCERT command to store the client certificate, not **gskkyman**.

In the examples later in this chapter, we assume that a key database has been set up. The procedure to set up a new key database (and stash file) is as follows:

- ▶ **Step 1.** Set up access to the **gskkyman** command from your UNIX shell. This is covered in *z/OS V1R2 System SSL Programming*, SC24-5901.
- ▶ **Step 2.** From the UNIX shell, enter the command **gskkyman**. Figure 10-7 shows the initial panel. This example shows how to create a new key database in the HFS. The database will be created in the subdirectory you entered the **gskkyman** command from. The password you enter here will be used to open the database in the future.

```
FOCAS @ SC63: />gskkyman

          IBM Key Management Utility

Choose one of the following options to proceed.

    1 - Create new key database
    2 - Open key database
    3 - Change database password

    0 - Exit program

Enter your option number: 1
Enter key database name or press ENTER for "key.kdb": example.kdb
Enter password for the key database.....>
Enter password again for verification.....>
Should the password expire? (1 = yes, 0 = no) [1]: 0

The database has been successfully created, do you want to continue to work with
the database now? (1 = yes, 0 = no) [1]: 0
```

Figure 10-7 Setting up a new key database in an HFS using **gskkyman**

Since the key database has a password, there must be a mechanism for a server to supply it to read the contents. This mechanism is implemented by using a stash file, which is a file using the same name as the key database, but with a suffix of .sth rather than .kdb. This file contains the key database password in encrypted form, and is created from the **gskkyman** panel.

- ▶ **Step 3.** Create the password stash file. This is shown in Figure 10-8 on page 211.

```

FOCAS @ SC63: />gskkyman

                IBM Key Management Utility

Choose one of the following options to proceed.

    1 - Create new key database
    2 - Open key database
    3 - Change database password

    0 - Exit program

Enter your option number: 2
Enter key database name or press ENTER for "key.kdb": example.kdb
Enter password for the key database.....>

                Key database menu

Current key database is /example.kdb

    1 - List/Manage keys and certificates
    2 - List/Manage request keys
    3 - Create new key pair and certificate request
    4 - Receive a certificate issued for your request
    5 - Create a self-signed certificate
    6 - Store a CA certificate
    7 - Show the default key
    8 - Import keys
    9 - Export keys
   10 - List all trusted CAs
   11 - Store encrypted database password

    0 - Exit program

Enter option number (or press ENTER to return to the parent menu): 11

The encrypted password has been stored in file /example.sth

Your request has completed successfully, exit gskkyman? (1 = yes, 0 = no) [0]: 1
FOCAS @ SC63: />ls -la example.*
-rw-----  1 HAIMO   SYS1      65080 May 15 17:57 example.kdb
-rw-----  1 HAIMO   SYS1         80 May 15 17:57 example.rdb
-rw-----  1 HAIMO   SYS1      129 May 15 18:06 example.sth
FOCAS @ SC63: />

```

Figure 10-8 Creation of the stash file using gskkyman

Note that after the stash file was created, the UNIX file attributes were displayed with the UNIX **ls** command. As you can see in Figure 10-8, the file attributes of the key database and the stash file are both “-rw-----” which means only the creator of the database (the user of the **gskkyman** command) can read and write to this file. You should use the UNIX **chmod** command to set the permission bits so that the server’s UNIX UID is able to read both the key database and the stash file. An example command to allow the owner read/write access and the owners group to have read access would be **chmod 640 example.***

10.6 Client certificates

An SSL/TLS enabled server *may* request that the client produce a digital certificate to verify the client's identity. The server must then validate the client certificate by checking the trusted CA hierarchy in its own key database, to ensure the digital signature on the certificate is from a trusted CA. The server does not make use of the client's public key contained in the certificate for communications; the request is for identification purposes only.

If the client passes a self-signed certificate (one that the client has generated and signed itself), then the server must check to ensure it has a copy of the same certificate in its key database and that the certificate is marked as trusted.

When client authentication is requested by the server, the server will be configured to authenticate to a particular level. These levels are:

Level 1 The server ensures the signer of the client's certificate is trusted by checking the trusted CA certificates that are in the server's keyring.

Level 2 The authentication requires that the client certificate be registered with RACF (or other SAF-compliant security product) and that it be in "TRUSTED" status. The RACF user ID that the certificate is associated with is that given in the ID() parameter of the RACFDCERT ID() ADD command when the client certificate was added to RACF. The CA that issued the client certificate must have a CA certificate connected to the server's keyring. Note that this level cannot be used if the z/OS server is using a key database created by using **gskkyman**.

Level 3 The authentication provides, in addition to level 1 and level 2 support, the capability to restrict access to the server based on the user ID returned from RACF. This level is implemented entirely in RACF, that is, a server only selects level 2 authentication, and if the appropriate profiles for the server are defined in RACF, the authentication level is upgraded to level 3. Note that this level cannot be used if the z/OS server is using a key database created by using **gskkyman**.

10.7 Server certificates

As discussed in "SSL protocol description" on page 186, the SSL/TLS protocol requires a server to supply a digital certificate to a client. The client must then validate the server certificate by checking the trusted CA hierarchy in its own key database, to ensure the digital signature on the certificate is from a trusted CA. Then the client can use the server's public key from the certificate to communicate the rest of the SSL handshake.

Important: To implement SSL in any form, you must have a server certificate available to the server and client. This is a prerequisite for implementing any client authentication that is discussed in this chapter.

If the server passes a self-signed certificate (one that the server has digitally signed itself), then the client must check to ensure it has a copy of the same certificate in its key database, and that the certificate is marked as "TRUSTED".

10.8 Self-signed certificates

The server or client certificate may be self-signed. This means that the digital signature on the certificate can only be verified by the public key given on the same certificate. The certificate is not authenticated by any Certificate Authority and must be taken at face value by the client or server receiving it.

The normal validation procedure for a certificate is still performed for a self-signed certificate. This means the receiver checks their key database for the Certificate Authority that signed the certificate, but as already mentioned, the CA is represented by the certificate received. Therefore, the certificate must have been previously received by some other means, and placed in the receiver's key database as a trusted certificate.

10.9 Obtaining certificates

This section shows the practical steps necessary to obtain digital certificates in a z/OS environment. If you choose not to use self-signed certificates, you will need to request your client/server certificates from a Certificate Authority (CA). That CA can be either an external organization such as VeriSign, or you can create a CA internally by generating a CA certificate yourself, and using that to sign other certificates. You can also generate self-signed certificates where the CA is the certificate itself, this is the simplest form of certificate usage.

In all the examples that follow, the server runs on z/OS under the RACF user ID "STC" and the end-user's RACF user ID is "FOCAS". The end-user's user ID is only needed when you are storing client certificates in RACF using RACDCERT.

Procedures for obtaining and storing self-signed certificates can be found in 10.9.1, "Self-signed certificates" on page 213. Procedures for obtaining and storing internal CA signed certificates can be found in 10.9.2, "Internal Certificate Authority (CA)" on page 229. Procedures for obtaining and storing external CA signed certificates can be found in 10.9.3, "External Certificate Authority (CA)" on page 234.

10.9.1 Self-signed certificates

The aim of this section is to show how to use the TSO RACDCERT command and the UNIX **gskkyman** command to store and use self-signed certificates. For the purposes of the examples, it is assumed the server is on z/OS and the client is not.

The procedure to use RACDCERT to generate and manage a self-signed server certificate is shown in "Self-signed server certificate RACDCERT procedure" on page 213. The procedure to use RACDCERT to import and manage a self-signed client certificate is shown in "Self-signed client certificate RACDCERT procedure" on page 218. The procedure to use **gskkyman** to generate and manage a self-signed server certificate is shown in "Self-signed server certificate gskkyman procedure" on page 221. The procedure to use **gskkyman** to import and manage a self-signed client certificate is shown in "Self-signed client certificate gskkyman procedure" on page 225.

Self-signed server certificate RACDCERT procedure

This procedure is basically the same for any z/OS server. In this example, we are generating and storing a certificate for use by a TN3270 server.

Once generated, the server certificate is placed in the server's RACF keyring and also exported to the client to be placed in the client's key database as a trusted CA certificate.

Here are the steps required:

- ▶ **Step 1.** Generate a self-signed certificate for the server.

```
//FOCAS1 JOB 'SET UP TN3270 CERT','PETER FOCAS',CLASS=A,MSGCLASS=X
//SERVCRT EXEC PGM=IKJEFT01,DYNAMNBR=30,REGION=4096K
/*
/* set up the TN3270 server certificate, and self-sign it.
/*
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *
RACDCERT ID(STC) GENCERT SUBJECTSDN(CN('ITSO.RALEIGH.IBM.COM') -
O('IBM Corporation') -
OU('ITSO Raleigh TN3270 Server') -
C('US')) -
WITHLABEL('TN3270 Server')
/*
```

Figure 10-9 Batch job to create self-signed server certificate

The ID(STC) parameter **I** associates the certificate being generated with the RACF user “STC”. This is the user ID that the server in our example is running under. Yours will probably be different. For an explanation of the rest of the RACDCERT parameters, see “Digital certificate field formats” on page 205.

Note that since there is no RACDCERT SIGNWITH parameter specified on the GENCERT command, the certificate will be digitally signed by the private key owned by the subject of the certificate. This is the definition of a self-signed certificate. Make sure the common name (CN) is the same as the host or domain name of the server.

- ▶ **Step 2.** Create a RACF keyring for the server.

```
//FOCAS1 JOB 'SET UP TN3270 CERT','PETER FOCAS',CLASS=A,MSGCLASS=X
//KEYRING EXEC PGM=IKJEFT01,DYNAMNBR=30,REGION=4096K
/*
/* Add a new Keyring to the TN3270 servers RACF ID (STC), then....
/* Add TN3270 server certificate to the user 'STC's keyring. the
/* Keyring name is from the TN3270 configuration statement as below
/* 'KEYRING SAF TN327ORing'
/*
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *
RACDCERT ID(STC) ADDRING(TN327ORing)
RACDCERT ID(STC) CONNECT(ID(STC) -
LABEL('TN3270 Server') -
RING(TN327ORing) -
DEFAULT -
USAGE(PERSONAL))
/*
```

Figure 10-10 Batch job to add a keyring for the self-signed certificate

Figure 10-10 shows the two steps necessary to create the keyring for the server:

- Create a new RACF keyring using the RACDCERT ADDRING command.
- Connect the self-signed servers certificate to the new keyring using the RACDCERT CONNECT command.

Note the RING parameter specifies the same ring name as what you would have configured into the server. In the TN3270 server, this is specified on the KEYRING SAF *ringname* statement, and on the FTP server it is on the KEYRING statement in FTP.DATA. The DEFAULT statement is needed because there may be more than one certificate in the keyring, and System SSL needs to know which certificate to pass to the client.

- ▶ **Step 3.** Export the self-signed server certificate to an MVS database.

```
//FOCAS1 JOB 'EXPORT SERVER CERT','PETER FOCAS',CLASS=A,MSGCLASS=X
//EXPORT EXEC PGM=IKJEFT01,DYNAMNBR=30,REGION=4096K
/*
/* Export the Self-signed Server certificate from the RACF database
/* in base-64 encoded format. This is then FTP'd to the TN3270
/* client so that it can verify the same certificate
/* when passed in the SSL exchange.
/*
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *
RACDCERT ID(STC) EXPORT(label('TN3270 Server')) -
FORMAT(CERTB64) DSN('FOCAS.RACDCERT.TN32CERT')
/*
```

Figure 10-11 Batch job to write the internal CA certificate to an MVS data set

Figure 10-11 shows the RACDCERT EXPORT command being used to export the self-signed server certificate to an MVS data set.

- ▶ **Step 4.** FTP the certificate exported to the MVS data set in step 3 to the client that will use it.

This step is not shown, since any FTP client will be able to perform this step. Note that in the example, the exported certificate in the MVS data set is in EBCDIC format. Therefore, the FTP must perform EBCDIC-to-ASCII translation if the client is on an ASCII host. The MVS data set will be FTPed to any client that needs to use that certificate to validate the same certificate when presented by a server in an SSL exchange. Depending on the number of clients in an enterprise, this may result in a large number of transfers. One way to reduce the number of file transfers to clients is for all clients to pick up their key database from a LAN drive.

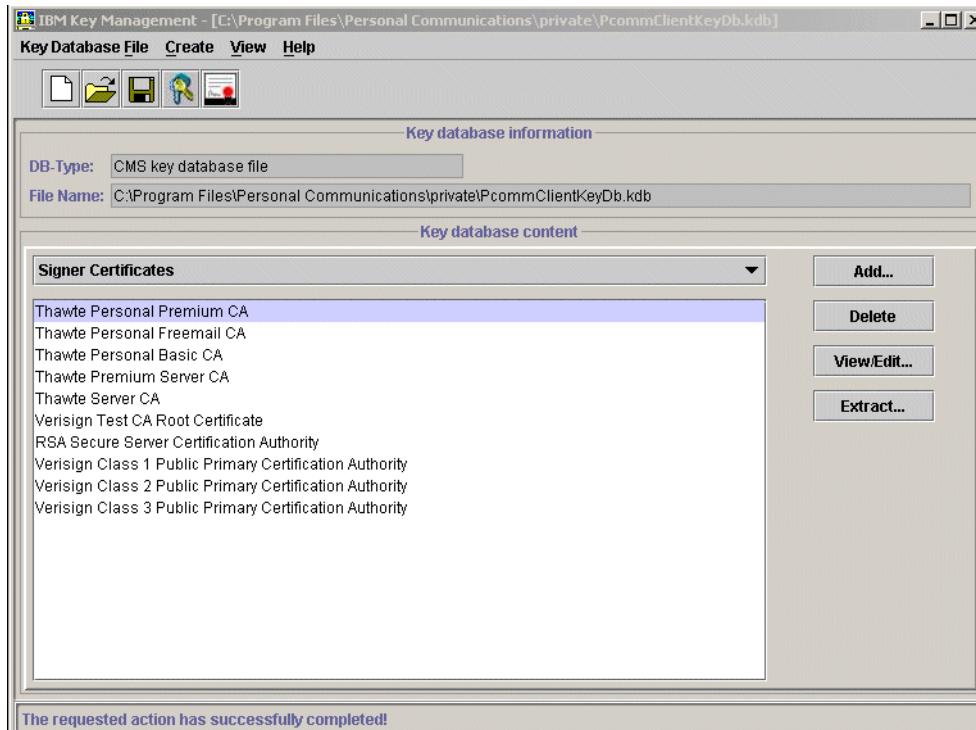


Figure 10-12 PCOMM client certificate management window

- **Step 5.** At the client, the certificate received from step 4 must be imported into the key database as a trusted certificate.

Depending on the type of client, there are a number of different ways to do this. In the case of a Windows PCOMM client (a TN3270 client) you select the **Certificate Management** or **Certificate Wizard** icon from the Utilities folder. This displays the window (after the key database file is opened) shown in Figure 10-12. You now import the certificate by clicking the **Add** button. Once the certificate is added, the window seen in Figure 10-13 on page 217 is the detail display from the certificate, showing the key size of 1024 (set by default in the RACDCERT GENCERT command), the certificate version, and the “Issued To” name the same as the “Issued By” (this is a self-signed certificate).

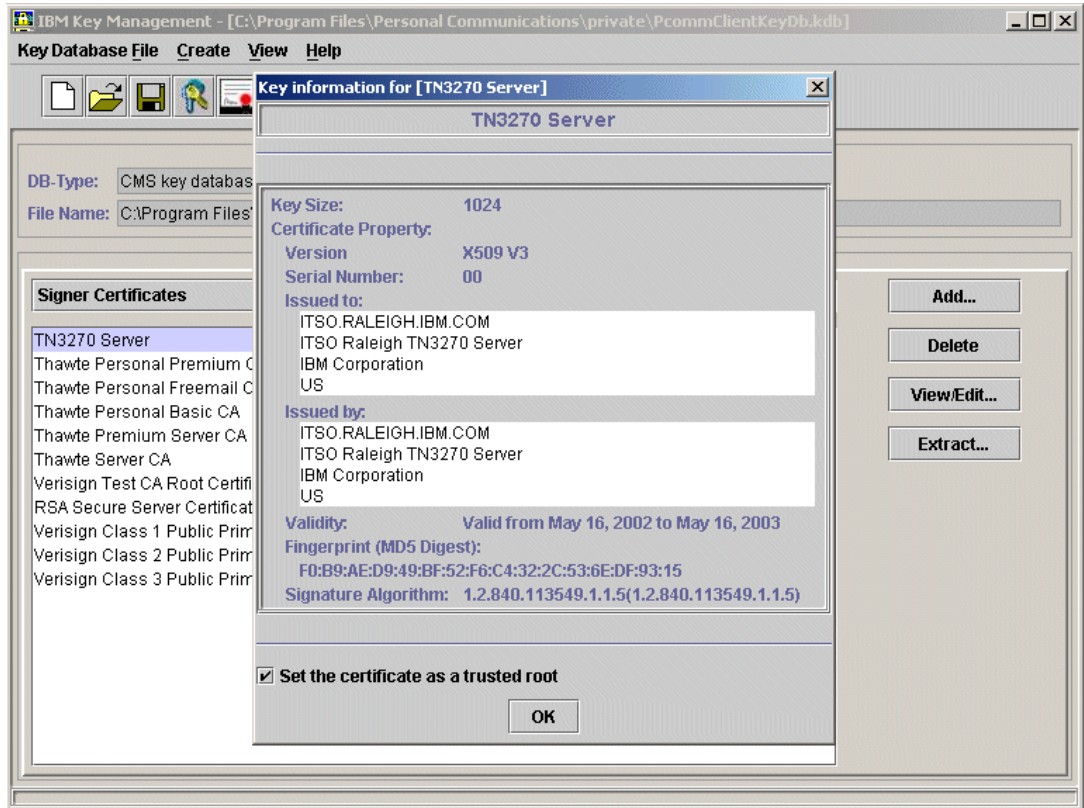


Figure 10-13 PCOMM client: display of imported self-signed server certificate

► **Step 6.** Test the client-to-server connection.

The PCOMM client was instructed to connect to the TN3270 server using SSL. Figure 10-14 on page 218 shows PCOMM displaying the server certificate (by clicking **Communication -> Security -> Server**), which shows that the certificate subject is the TN3270 server, and the certificate issuer is the same. This is the self-signed server certificate set up in step 1.

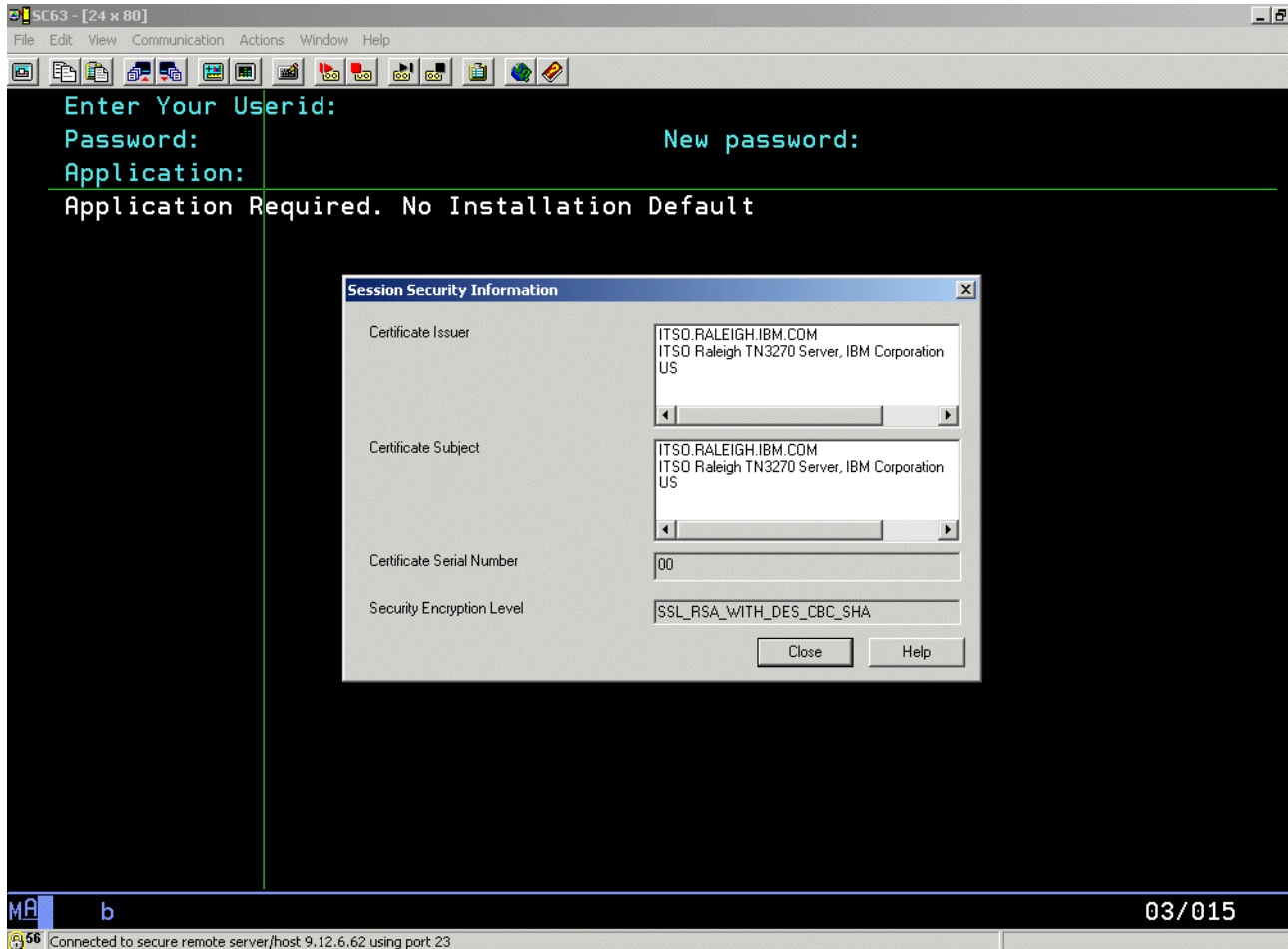


Figure 10-14 PCOMM client: server certificate being used and signer's details

While this discussion showed the certificate being generated for a TN3270 server, followed by an export to the client and placement in the client's key database as a trusted certificate, the procedure for any client/server is basically the same.

Self-signed client certificate RACDCERT procedure

A client certificate must be added to RACF and associated with the appropriate RACF user ID using the RACFDCERT ID(clients-user ID) ADD.... statement. The client certificate's CA must be connected to the server's RACF keyring using the RACFDCERT ID(servers-user ID) CONNECT. The basic procedure to follow is:

- ▶ **Step 1.** Get the client certificate. For a self-signed certificate, this is normally generated at the client end.

Since different client programs have different ways to generate a client certificate, this should be thought of as a generic example. Most clients will have some way to generate a certificate. In the case of PCOMM, which provides a TN3270 client, you use the Windows menu (click **Start -> Programs -> IBM Personal Communications -> Utilities -> Certificate Management**) to open the client's key database. Then you click **Create -> New Self-signed Certificate** to generate the certificate. See Figure 10-15 on page 219 for an example of a self-signed client certificate that has just been generated by PCOMM into the client key database.

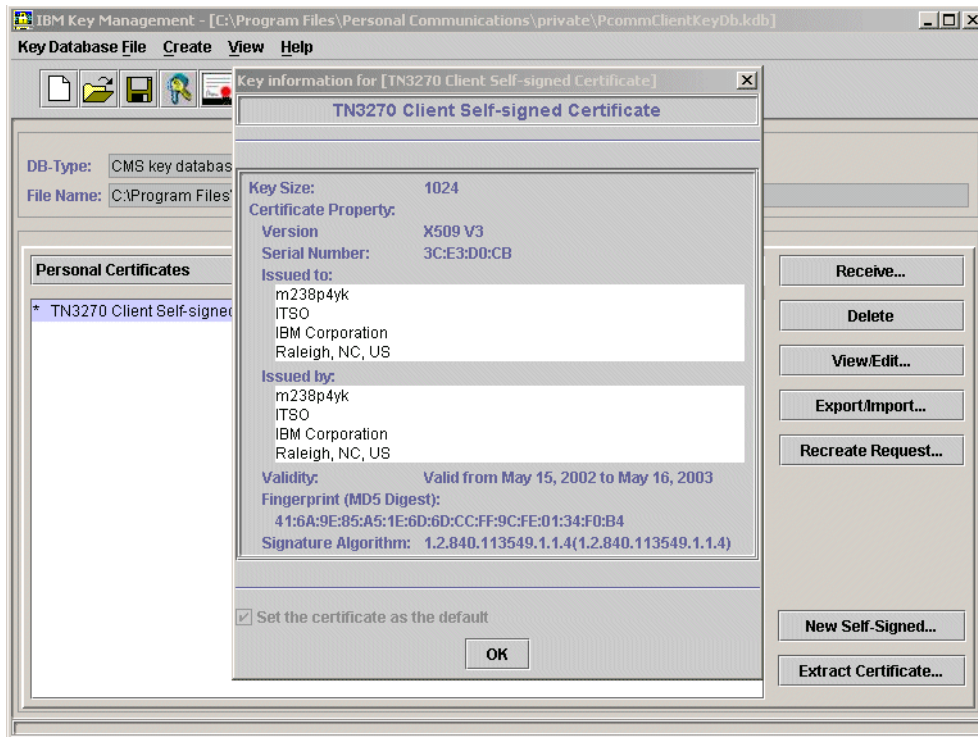


Figure 10-15 PCOMM client: newly added self-signed client certificate

- ▶ **Step 2.** Export the certificate from the client's key database to a certificate file.
 Export the client certificate to a file. Most certificate management utilities allow the export of a certificate in at least two formats. The most common is Base64-encoded ASCII, but there is also the binary DER format. The choice depends on what your certificate management utility at the server end can use as an import format. We will use Base64 ASCII format. At the lower right-hand side of the window shown in Figure 10-15 is the Extract Certificate... button. This is used to write a certificate to a data set. The window that is presented is shown in Figure 10-16 on page 220. Note the Data Type field specifies Base64-encoded ASCII and that the certificate will be written to the cert.arm file.

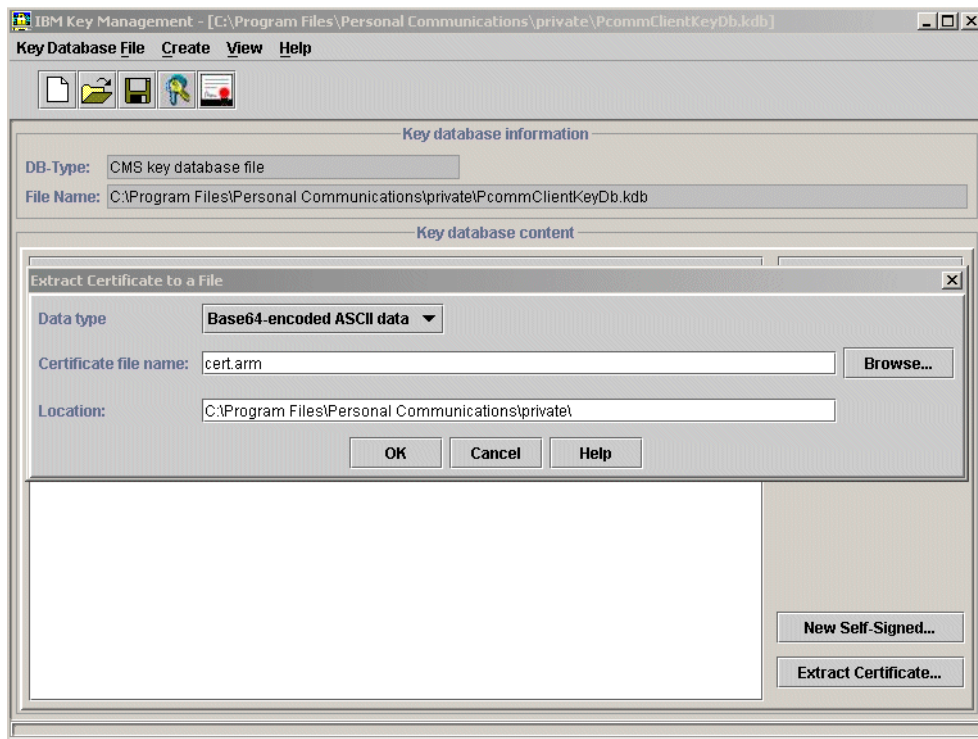


Figure 10-16 Exporting a client certificate to a file

- ▶ **Step 3.** FTP the certificate file from the client to the z/OS server side.

This does not need to be shown here. You just need to FTP the certificate file created in step 2 (cert.arm in this example) to an MVS data set at the server side as a text file.

- ▶ **Step 4.** Add the client certificate into the RACF database and associate with an end user.

Figure 10-17 shows the batch job used to add the client certificate to the RACF database and associate it with the RACF user ID “FOCAS” (the owner of the client certificate) with the ID() parameter. The RACDCERT ADD statement specifies the MVS data set name that contains the client certificate; this was the data set name that was created by the FTP in step 3. The certificate is set to TRUSTED status, which means that any certificate that is signed with this certificate will pass authentication.

```
//FOCAS1 JOB 'EXPORT SERVER CERT','PETER FOCAS',CLASS=A,MSGCLASS=X
//CLIENT EXEC PGM=IKJEFT01,DYNAMNBR=30,REGION=4096K
/**
/** Import the Self-signed Client certificate into the RACF database
/** This was FTP's from the workstation TN3270 client, that
/** generated it with the local PCOMM utility. It must be
/** imported as a trusted CA certificate
/**
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *
RACDCERT ID(FOCAS) ADD('FOCAS.RACDCERT.CLIENT.CERT') -
TRUST WITHLABEL('TN3270 client certificate PF')
/*
```

Figure 10-17 Batch job to add a client certificate into the RACF database as TRUSTED

- **Step 5.** Connect the client certificate to the server's RACF keyring.

Figure 10-18 shows the batch job used to connect the client certificate, added in step 4, to the server's keyring. In the RACDCERT CONNECT statement, the keyring name is whatever is coded in the server's configuration file (in TN3270's case it is specified on the KEYRING SAF statement). This assumes the keyring is already set up (probably because you have the server's certificate there). If the ring is not set up, you must create it before this job is run with the RACDCERT ID(STC) ADDRING(TN3270Ring) command (assuming the server's RACF ID is "STC" and keyring name is "TN3270Ring").

```

//FOCAS1  JOB 'EXPORT SERVER CERT','PETER FOCAS',CLASS=A,MSGCLASS=X
//CLIENT  EXEC PGM=IKJEFT01,DYNAMNBR=30,REGION=4096K
//*
//* Import the Self-signed Client certificate into the RACF database
//* This was FTP's from the workstation TN3270 client, that
//* generated it with the local PCOMM utility. It must be
//* imported as a trusted CA certificate
//*
//SYSTSPRT DD  SYSOUT=*
//SYSTSIN  DD  *
RACDCERT ID(STC) CONNECT(ID(FOCAS)          -
                        LABEL('TN3270 client certificate PF') -
                        RING(TN3270Ring)          -
                        USAGE(PERSONAL))
/*

```

Figure 10-18 Batch job to import client certificate into RACF

After these steps have been taken (assuming you have followed the steps to obtain and store the server's certificate) and both the client and server are configured with the appropriate parameters for client authentication, the connection can be made.

Self-signed server certificate gskkyman procedure

It is assumed that you have set up a key database and produced a stash file for the server as discussed in 10.5, "gskkyman command use" on page 210, that you have set the correct UNIX permission bits so that the server can read the files and that the database is named tn32v1r2.kdb. Once the database is created, you generate a self-signed certificate into it, and set it as the default certificate. The self-signed certificate must then be exported to the client workstation, where it is imported into the client's key database as a trusted CA certificate.

Figure 10-19 on page 222 shows gskkyman being used to open the existing database in preparation for generating the certificate.

- **Step 1.** Open the key database using gskkyman.

```

FOCAS @ SC63:/>gskkyman

          IBM Key Management Utility

Choose one of the following options to proceed.

    1 - Create new key database
    2 - Open key database
    3 - Change database password

    0 - Exit program

Enter your option number: 2
Enter key database name or press ENTER for "key.kdb": tn32v1r2.kdb
Enter password for the key database.....>

          Key database menu

Current key database is /tn32v1r2.kdb

    1 - List/Manage keys and certificates
    2 - List/Manage request keys
    3 - Create new key pair and certificate request
    4 - Receive a certificate issued for your request
    5 - Create a self-signed certificate
    6 - Store a CA certificate
    7 - Show the default key
    8 - Import keys
    9 - Export keys
   10 - List all trusted CAs
   11 - Store encrypted database password

    0 - Exit program

Enter option number (or press ENTER to return to the parent menu):

```

Figure 10-19 Opening the key database

Now that the key database is opened, you are presented with a list of options. Option 5 creates a self-signed certificate by prompting for details. The generation of the self-signed certificate is shown in Figure 10-20 on page 223, which carries on from Figure 10-19 by selecting option 5.

- ▶ **Step 2.** Create the self-signed certificate and export it to an HFS file.

```

Enter option number (or press ENTER to return to the parent menu): 5
Enter version number of the certificate to be created (1, 2, or 3) [3]:
Enter a label for this key.....> TN3270 gskkyman certificate
Select desired key size from the following options (512):
    1: 512
    2: 1024
Enter the number corresponding to the key size you want: 2
Enter certificate subject name fields in the following.
    Common Name (required).....> ITSO.RALEIGH.IBM.COM
    Organization (required).....> IBM
    Organization Unit (optional).....> ITSO Raleigh TN3270 Server
    City/Locality (optional).....> Raleigh
    State/Province (optional).....> NC
    Country Name (required 2 characters)..> US
Enter number of valid days for the certificate [365]:
Do you want to set the key as the default in your key database? (1 = yes, 0 = no
) [1]: 1
Do you want to save the certificate to a file? (1 = yes, 0 = no) [1]: 1
Should the certificate binary data or Base64 encoded ASCII data be saved? (1 = A
SCII, 2 = binary) [1]:
Enter certificate file name or press ENTER for "cert.arm": tn32v1r2.arm

Please wait while self-signed certificate is created...

Your request has completed successfully, exit gskkyman? (1 = yes, 0 = no) [0]: 1
FOCAS @ SC63:/>

```

Figure 10-20 Generating a self-signed certificate using gskkyman

Figure 10-20 shows the dialog between the user and gskkyman to set up a self-signed certificate. The key size is requested, then details about the certificate subject are entered, whether you want to set the certificate as the default certificate and whether you want to produce an exported ASCII file of the certificate. This exported certificate is what is FTPed to the client for importation into the client's key database. In this example, the exported file is called tn32v1r2.arm. The contents of the exported file are shown in Figure 10-21.

```

FOCAS @ SC63:/>cat tn32v1r2.arm
-----BEGIN CERTIFICATE-----
MIICczCCAdygAwIBAgIEP00wjANBgkqhkiG9w0BAQFADBMQswCQYDQVQQGEwJV
UzELMAkGA1UECBMxMDEwMDAwMDA0BjNVBAcTB1JhbGVpZ2gxDDAKBgNVBAoTA01CTTEj
MCEGA1UECmMaSVRRTTyBSYWx1aWdoIFROMzI3M0BjNVBAcTB1JhbGVpZ2gxDDAKBgNV
U08uUkFMRU1HSC5JQk0uQ09NMB4XDTAyMDUxNTEzMTM1MFOXDTAzMDUxNjEzMTM1
MFowfjELMAkGA1UEBhMCVVMxMDEwMDAwMDA0BjNVBAcTB1JhbGVpZ2gxDDAKBgNV
MQwwCgYDQVQQEwNjQkOxIzAhBgNVBAsTGk1UU08gUmFsZWlnaCBUTjMyNzAgU2Vy
dmVybWROGwYDQVQQDEXRJVFNPLlJBTEVJR0guSUJNLkNPTTcBnzANBgkqhkiG9w0B
AQEFAA0BjQAwgYkCgYEAvuHpLXympFCoT1Q3jZ5E+EveDyued1RUo+BgCWOEErB/
6rKLrn1VwEU8w/nnTyApBW19IEbITrJ3YFGa4tIJV11eCQpGj5yQJNyPj6MIYOzv
9xDD8TgJu61zciJWLN6cnC7sygHiC+gEhCVAs+LR2wspzf0v8ebQZpujQDr2ZT0C
AwEAATANBgkqhkiG9w0BAQFADBMQswCQYDQVQQEwNjQkOxIzAhBgNVBAsTGk1UU08gUmFsZWlnaCBUTjMyNzAgU2Vy
F+eHCwDSHzo6JI1q21/Jk1bh/A4d9/ftN0rHOTS4rD1/U/izJR7tMSbJ/7kSAeZj
NXJDgQOIjpkMyZS8FQR9+BCRTD9EhDmaJGzxPxQ7U0F9Kth0c87NkMka06BNhUm1
5DzQ9Vjiag==
-----END CERTIFICATE-----
FOCAS @ SC63:/>

```

Figure 10-21 An exported certificate from the gskkyman utility

- ▶ **Step 3.** FTP the file exported in step 3 to the client that will be using it.
Any FTP client at the workstation can be used for this transfer. Ensure the transfer type is “text” and that you can view the file at the workstation as a text file.

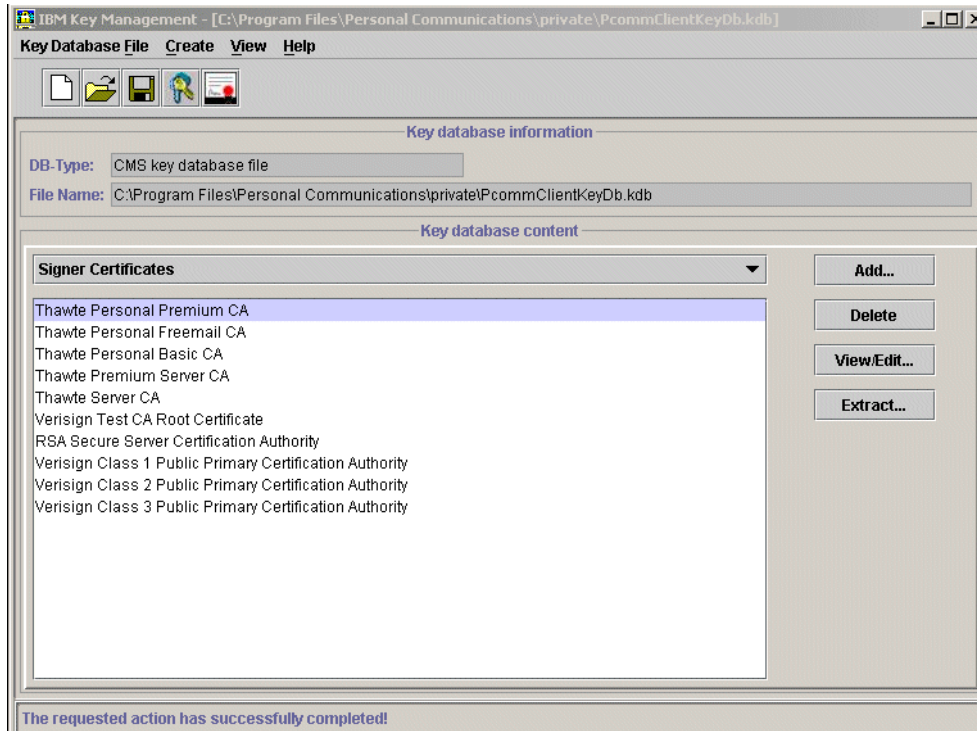


Figure 10-22 PCOMM client certificate management window

- ▶ **Step 4.** Import the certificate into the client’s key database.
At the client, the certificate received from step 4 must be imported into the key database as a trusted certificate. Depending on the type of client, there are a number of different ways to do this. In the case of a Windows PCOMM client (a TN3270 client), you select the **Certificate Management** or **Certificate Wizard** icon from the Utilities folder. This displays the window (after the key database file is opened) shown in Figure 10-22. You now import the certificate by clicking the **Add** button. Once the certificate is added, the window shown in Figure 10-23 on page 225 is the detail display from the certificate, showing the key size of 1024 (set by **gskkyman** when creating the certificate in Figure 10-20 on page 223), the certificate version, and the “Issued To” name the same as the “Issued By” (this is a self-signed certificate).

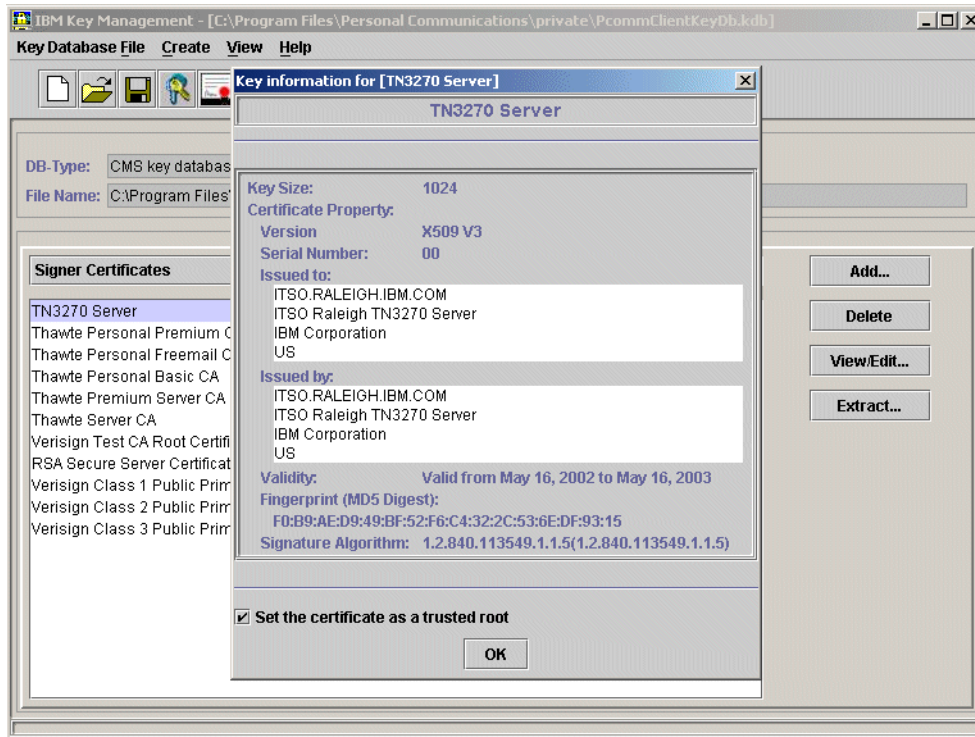


Figure 10-23 PCOMM client: display of imported self-signed server certificate

Now the client should be able to connect to the server. Once the server passes its certificate to the client during the SSL exchange, the client will be able to validate it using the same certificate that is now stored in the client's key database as a CA certificate.

Self-signed client certificate gskkyman procedure

It is assumed that you have set up a key database (tn32v1r2.kdb) and the server's certificate is in the key database. The steps to implement client authentication in a gskkyman environment are basically the same as in the RACDCERT environment, except the certificates are stored in an HFS key database rather than the RACF database. These steps are:

- ▶ **Step 1.** Get the client certificate. For a self-signed certificate, this is normally generated at the client end.

Since different client programs have different ways to generate a client certificate, this should be thought of as a generic example. Most clients will have some way to generate a certificate. In the case of PCOMM, which provides a TN3270 client, you use the Windows menu (click **Start > Programs -> IBM Personal Communications -> Utilities -> Certificate Management**) to open the client's key database. Then you click **Create -> New Self-signed Certificate** to generate the certificate. See Figure 10-24 on page 226 for an example of a self-signed client certificate that has just been generated by PCOMM into the client key database.

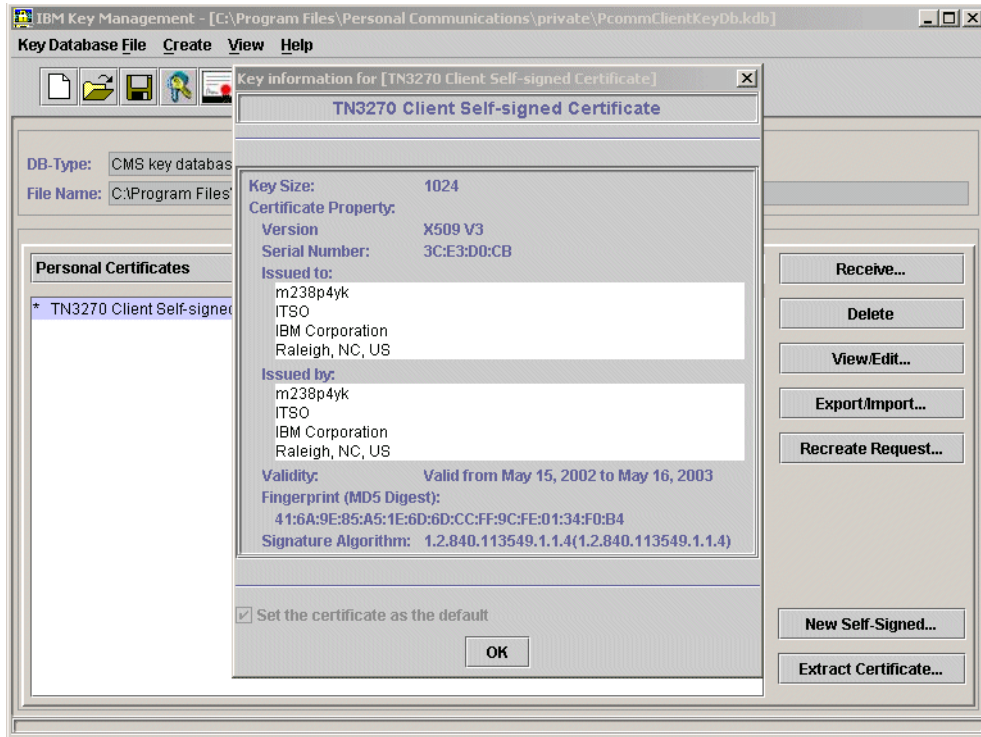


Figure 10-24 PCOMM client: newly added self-signed client certificate

► **Step 2.** Export the client certificate to a file.

Most certificate management utilities allow the export of a certificate in at least two formats. The most common is Base64-encoded ASCII, but there is also the binary DER format. The choice depends on what your certificate management utility at the server end can use as an import format. We will use Base64 ASCII format. At the lower right-hand side of the window shown in Figure 10-24 is the Extract Certificate... button. This is used to write a certificate to a file. The window that is presented is shown in Figure 10-25 on page 227. Note the Data Type field specifies Base64-encoded ASCII and that the certificate will be written to the cert.arm file.

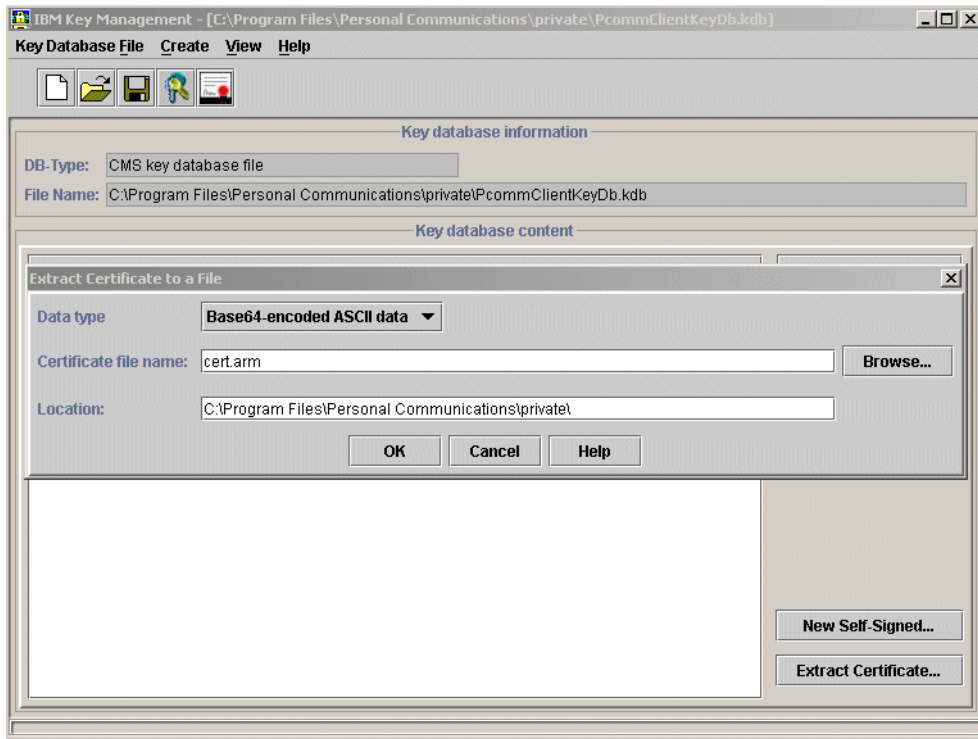


Figure 10-25 Exporting a client certificate to a file

- ▶ **Step 3.** FTP the certificate file from the client to the z/OS server side.
This does not need to be shown here. You just need to FTP the certificate file created in step 2 (cert.arm in this example) to an HFS data set at the server side as a text file.
- ▶ **Step 4.** Add the client certificate into the server's HFS key database using gskkyman.
Figure 10-26 on page 228 shows **gskkyman** being used to open the existing database in preparation for importing the client's certificate.

```

FOCAS @ SC63:/>gskkyman

          IBM Key Management Utility

Choose one of the following options to proceed.

    1 - Create new key database
    2 - Open key database
    3 - Change database password

    0 - Exit program

Enter your option number: 2
Enter key database name or press ENTER for "key.kdb": tn32v1r2.kdb
Enter password for the key database.....>

          Key database menu

Current key database is /tn32v1r2.kdb

    1 - List/Manage keys and certificates
    2 - List/Manage request keys
    3 - Create new key pair and certificate request
    4 - Receive a certificate issued for your request
    5 - Create a self-signed certificate
    6 - Store a CA certificate
    7 - Show the default key
    8 - Import keys
    9 - Export keys
   10 - List all trusted CAs
   11 - Store encrypted database password

    0 - Exit program

Enter option number (or press ENTER to return to the parent menu):

```

Figure 10-26 Opening the key database

Now that the key database is opened, you are presented with a list of options. Choose option **6 Store a CA certificate** to receive a certificate and store it as a trusted Certificate Authority certificate.

```

Key database menu

Current key database is /tn32v1r2.kdb

 1 - List/Manage keys and certificates
 2 - List/Manage request keys
 3 - Create new key pair and certificate request
 4 - Receive a certificate issued for your request
 5 - Create a self-signed certificate
 6 - Store a CA certificate
 7 - Show the default key
 8 - Import keys
 9 - Export keys
10 - List all trusted CAs
11 - Store encrypted database password

 0 - Exit program

Enter option number (or press ENTER to return to the parent menu): 6
Enter certificate file name or press ENTER for "cert.arm": cert.arm
Enter a label for this key.....> TN3270 Client Cert for User1

Please wait while certificate is stored...

Your request has completed successfully, exit gskkyman? (1 = yes, 0 = no) [0]: 1
FOCAS @ SC63:/>
===>

```

Figure 10-27 Adding a self-signed client certificate as a CA certificate using gskkyman

Figure 10-27 shows the dialog between the user and gskkyman to add the client certificate as a CA certificate. The client certificate file in the example is named “cert.arm” and was placed there in the FTP transfer in step 3.

Now the client should be able to connect to the server (assuming you have followed the steps to obtain and store the server’s certificate). The server will be able to validate the client certificate with its copy of the client’s certificate in the server’s key database.

10.9.2 Internal Certificate Authority (CA)

One possibility in setting up your certificate management scheme is to set up as a Certificate Authority. This means that you will be signing digital certificates for other entities, and anybody who uses the certificates that you sign will have to have a copy of your certificate in their key databases as a trusted CA.

You might choose to be a Certificate Authority if you have multiple SSL/TLS enabled servers in your system. When you have more than one server with its own self-signed certificate, each certificate must be exported to the clients that will use them. Therefore, if you had an FTP server, a TN3270 server and an LDAP server, each using self-signed certificates and all being used from one workstation, that workstation would need all three certificates in its key database. With an internal CA, you can sign each of the server’s certificates, and export the one certificate (the internal CA certificate) to the clients. Please note that this assumes that a client’s key database can be shared between client programs. This is mostly not the case, but a saving can still be made in that only the one key needs to be distributed.

In this section, we cover setting up an internal CA, and creating and signing a server certificate with that CA certificate. Only one example is given, that is signing a server certificate and using RACDCERT for storing the certificates. You will see that the process is similar to the self-signing process described in “Self-signed certificates” on page 213, except that the certificate being distributed to the clients is the CA certificate, not the server certificate.

Internal-CA signed server certificate RACDCERT procedure

This procedure is basically the same for any z/OS server. In this example, we are producing a certificate for use by a TN3270 server. This certificate is needed by TN3270 clients using SSL. For the client to validate the server certificate, the internal CA certificate will be needed at the client.

- **Step 1.** Create a self-signed certificate for the local (internal) CA.

```
//CERTAUTH EXEC PGM=IKJEFT01,DYNAMNBR=30,REGION=4096K
/*
/* Add the top-level self-signed certificate for the certificate
/* authority (ourselves)
/*
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *
racdcert certauth gencert -
subjectsdn( o('IBM Corporation') -
ou('ITSO Certificate Authority') -
C('US')) -
WITHLABEL('ITSO Certificate Authority')
/*
```

Figure 10-28 Batch job to create internal CA certificate in RACF database

- **Step 2.** Generate a certificate for the server.

```
//SERVCRT EXEC PGM=IKJEFT01,DYNAMNBR=30,REGION=4096K
/*
/* set up the TN3270 server certificate, and sign it with the
/* self-signed certificate-authority certificate set up previously
/*
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *
RACDCERT ID(STC) GENCERT SUBJECTSDN(CN('ITSO.RALEIGH.IBM.COM') -
O('IBM Corporation') -
OU('ITSO Raleigh TN3270 Server') -
C('US')) -
WITHLABEL('TN3270 Server') -
SIGNWITH(CERTAUTH LABEL('ITSO Certificate Authority'))
/*
```

Figure 10-29 Batch job to create server certificate and sign with internal CA certificate

Note that the RACDCERT SIGNWITH parameter specifies the LABEL of the internal CA certificate we set up in step 1. This indicates that the server certificate should be digitally signed with the internal CA’s private key. The ID(STC) parameter is used in this example, because the TN3270 server, for whom the certificate is being generated, is associated with RACF user ID “STC”. Make sure the common name (CN) is the same as the host or domain name of the server.

► **Step 3.** Create a RACF keyring for the server.

```
//KEYRING EXEC PGM=IKJEFT01,DYNAMNBR=30,REGION=4096K
/**
/** Add a new Keyring to the TN3270 servers RACF ID (STC), then....
/** Add TN3270 server certificate to the user 'STC's keyring. the
/** Keyring name is from the TN3270 configuration statement as below
/** 'KEYRING SAF TN327ORing'
/**
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *
RACDCERT ID(STC) ADDRING(TN327ORing)
RACDCERT ID(STC) CONNECT(CERTAUTH -
                        LABEL('ITSO Certificate Authority') -
                        RING(TN327ORing) -
                        USAGE(CERTAUTH))
RACDCERT ID(STC) CONNECT(ID(STC) -
                        LABEL('TN3270 Server') -
                        RING(TN327ORing) -
                        DEFAULT -
                        USAGE(PERSONAL))
/*
```

Figure 10-30 Batch job to add a keyring

Figure 10-30 shows the three steps necessary to create the keyring for the server:

- Create a new RACF keyring using the RACDCERT ADDRING command
- Connect the internal CA certificate to the new keyring using the RACDCERT CONNECT command
- Connect the server's certificate (which was signed by the internal CA certificate) to the new keyring using the RACDCERT CONNECT command

► **Step 4.** Export the internal CA certificate to an MVS data set.

```
//EXPORT EXEC PGM=IKJEFT01,DYNAMNBR=30,REGION=4096K
/**
/** Export the Self-signed Certificate Authority certificate from the
/** RACF database in base-64 encoded format. This is then FTP'd to
/** the TN3270 client so that it can verify the TN3270 server's
/** certificate when passed in the SSL exchange.
/**
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *
RACDCERT CERTAUTH EXPORT(label('ITSO Certificate Authority')) -
                        FORMAT(CERTB64) DSN('FOCAS.RACDCERT.SERVCERT')
/*
```

Figure 10-31 Batch job to write the internal CA certificate to an MVS data set

Figure 10-31 shows the RACDCERT EXPORT command being used to export the internal CA certificate to an MVS data set. The MVS data set will be ASCII FTPed to any client that needs to use that certificate to validate a server (in this case a TN3270 client). One way to reduce the number of file transfers to clients is for them to pick up their key databases from a LAN drive, if practicable.

- ▶ **Step 5.** FTP the certificate exported in step 4 to the client that will use it.
This step is not shown, since any FTP client will be able to perform this step.

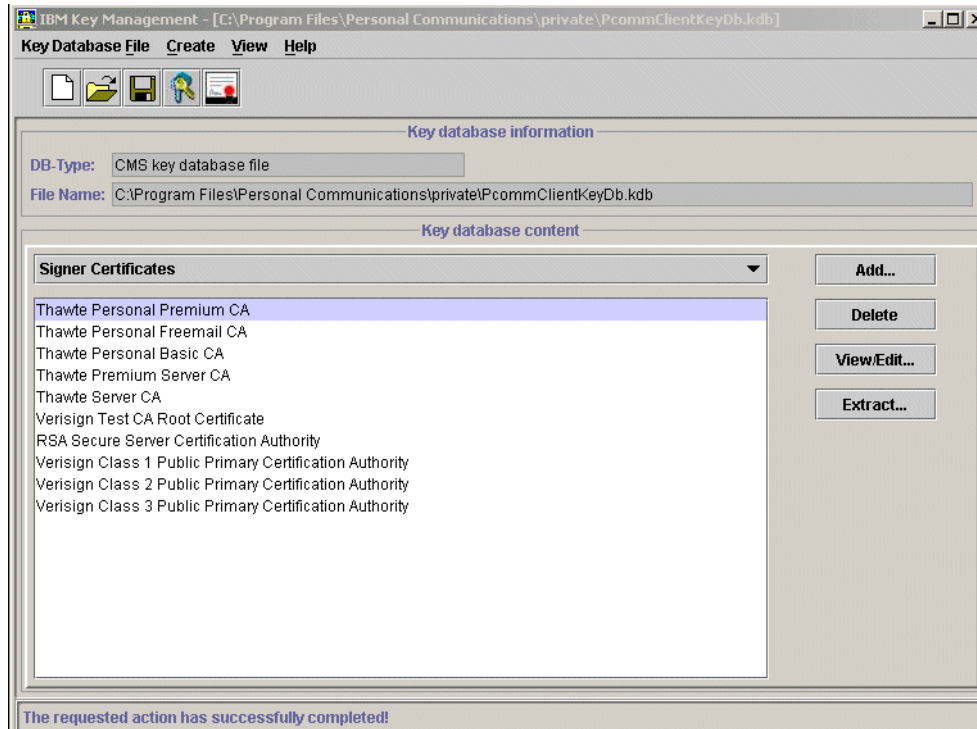


Figure 10-32 PCOMM client certificate management window

- ▶ **Step 6.** At the client, the certificate received from step 5 must be imported into the key database as a trusted certificate.

Depending on the type of client, there are a number of different ways to do this. In the case of a Windows PCOMM client (a TN3270 client) you select the **Certificate Management** or **Certificate Wizard** icon from the Utilities folder. This displays the window (after the key database file is opened) in Figure 10-32. You now import the certificate using the **Add** button. Once the certificate is added, the window shown in Figure 10-33 on page 233 is the detail display from the certificate, showing the key size of 1024 (set by default in the RACDCERT GENCERT command), the certificate version, and the “Issued To” name the same as the “Issued By” (this is a self-signed certificate for a CA).

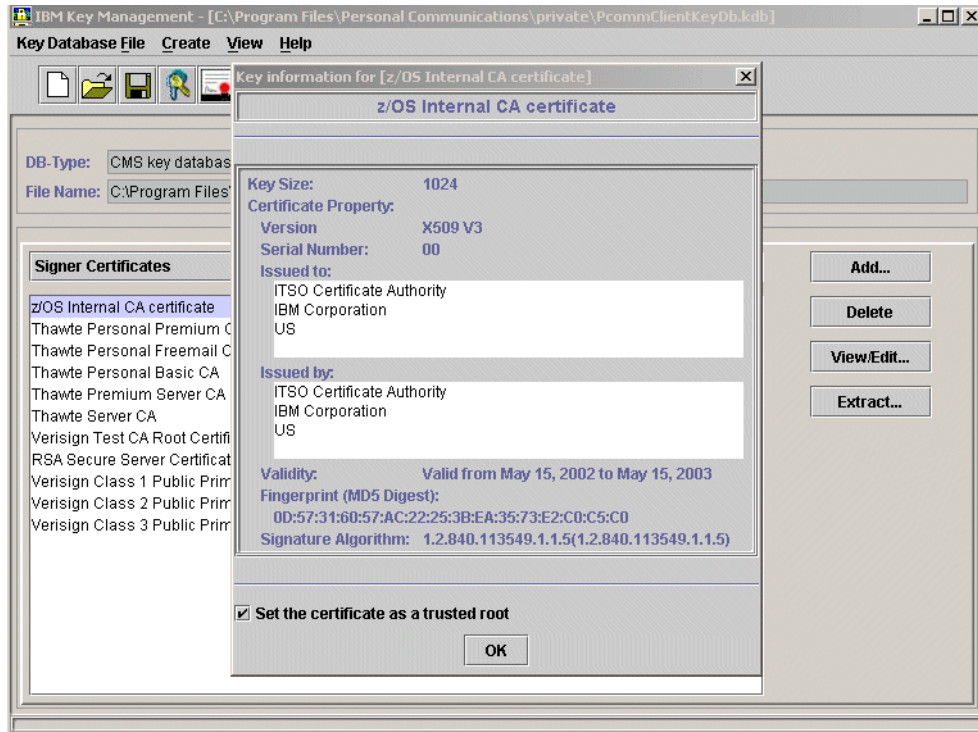


Figure 10-33 PCOMM client: display of imported server certificate

► **Step 7.** Test the client to server connection.

The PCOMM client was instructed to connect to the TN3270 server using SSL. Figure 10-34 on page 234 shows PCOMM displaying the server certificate (by clicking **Communication -> Security -> Server**), which shows that the certificate subject is the TN3270 server, and the certificate issuer is the internal CA set up in step 1.

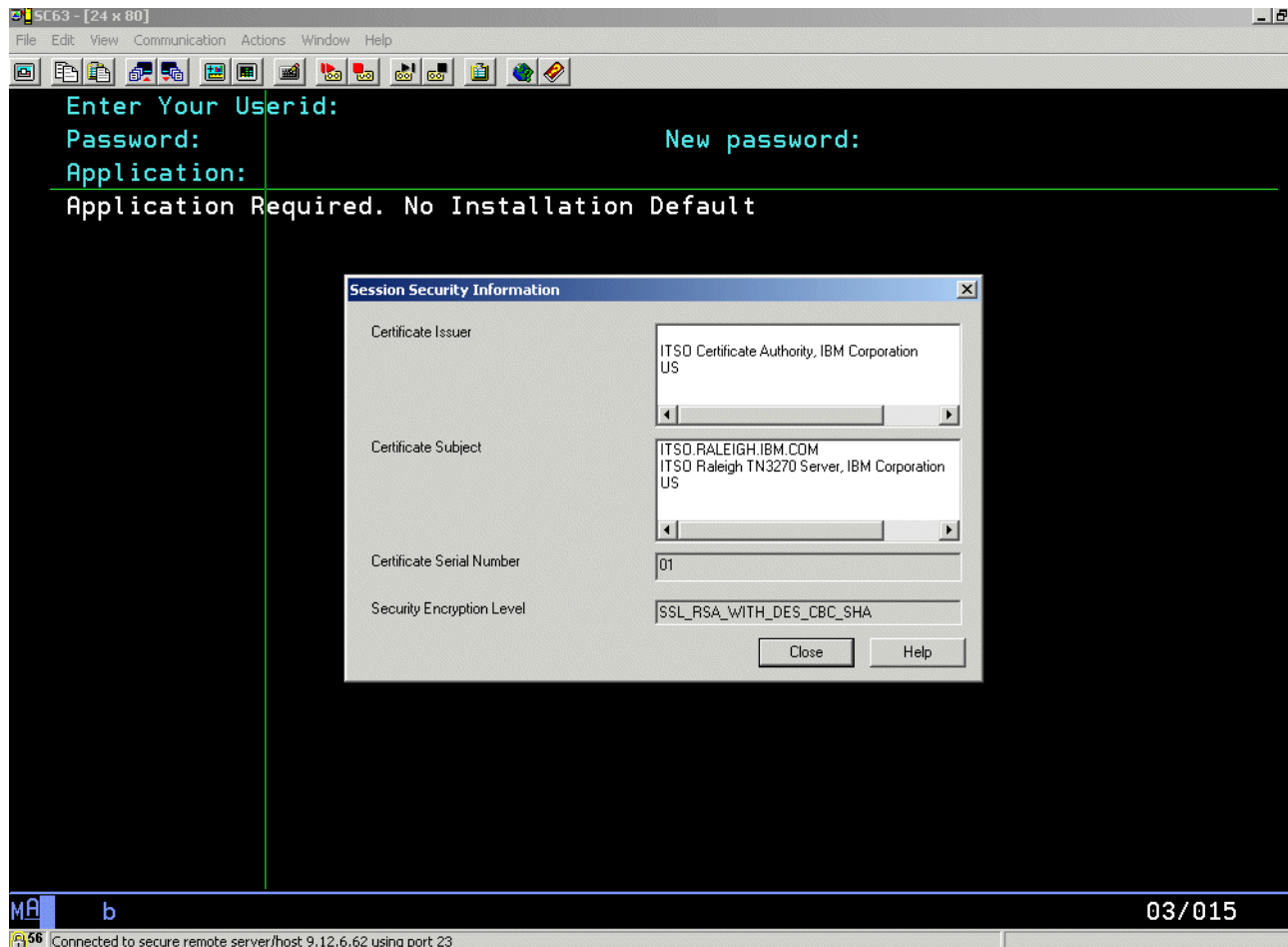


Figure 10-34 PCOMM client: server certificate and signer's details

While this discussion showed certificates being generated for a TN3270 server, and the internal CA certificate being exported to the client, then placed in the client's key database as a trusted certificate, the procedure for any client/server is basically the same.

10.9.3 External Certificate Authority (CA)

The aim of this section is to show how to use the TSO RACDCERT command and the UNIX **gskkyman** command to store and use CA-signed certificates. For the purposes of the examples, it is assumed that the server is on z/OS and the client is not.

The following step-by-step examples are generic in nature. They can be used to create an HFS key database or RACF certificate/keyring for the IBM HTTP Server for z/OS, the TN3270 server, or other servers that are SSL enabled.

The procedure to use RACDCERT to request and manage a CA-signed server certificate is shown in "External CA-signed server certificate RACDCERT procedure" on page 235.

The procedure to use RACDCERT to request and manage a CA-signed client certificate is shown in "External CA-signed client certificate RACDCERT procedure" on page 241. The procedure to use **gskkyman** to request and manage a CA-signed server certificate is shown in "External CA-signed server certificate gskkyman procedure" on page 236. The procedure to use **gskkyman** to request and manage a CA-signed client certificate is shown in "External CA-signed client certificate gskkyman procedure" on page 247.

External CA-signed server certificate RACDCERT procedure

This section presents the steps required to implement the SSL environment for IBM HTTP Server with a server certificate signed by a public CA. A similar procedure can be used for other SSL-enabled application servers. The assumption in the examples is that the Web server's RACF user ID is "WEBSRV".

- ▶ **Step 1.** Generate a self-signed certificate.

We will use this certificate as a base for the certificate request we will be creating.

```
RACDCERT ID(WEBSRV) GENCERT
          SUBJECTDSN(CN('itso.raleigh.ibm.com')
                    O('IBM Corporation')
                    OU('ITSO Raleigh Webserver')
                    C('US'))
          WITHLABEL('Web Server Certificate')
```

Make sure the common name (CN) is the same as the host or domain name of the server.

- ▶ **Step 2.** Create a certificate request for the CA.

The certificate request will be stored in an MVS data set named BOCHE.WEBSERV.GENREQ.

```
RACDCERT ID(WEBSRV) GENCERT
          GENREQ(LABEL('Web Server Certificate'))
          DSN('BOCHE.WEBSERV.GENREQ')
```

This certificate request needs to be sent to the Certificate Authority. The format of the request is Base64-encoded text. The data set can be transmitted to a PC with FTP and pasted into the appropriate field in the certificate request. Alternatively, cutting and pasting between a host emulator window and the Web browser can be used.

- ▶ **Step 3.** Store the returned certificate into an MVS data set.

The CA usually returns the certificate using e-mail or similar means. The certificate is in Base64-encoded text format. Use FTP to transfer the certificate received from the CA into an MVS data set named, for instance, BOCHE.WEBSERV.CERT.

- ▶ **Step 4.** Replace the self-signed certificate with the certificate received from and signed by the CA.

```
RACDCERT ID(WEBSRV)
          ADD('BOCHE.WEBSERV.CERT')
          WITHLABEL('Web Server Certificate')
```

- ▶ **Step 5.** Create a keyring for the server.

This keyring must not already exist for this user. Keyring names becomes names of RACF profiles in the DIGTRING class, and can contain only characters that are allowed in RACF profile names. Although asterisks are allowed in keyring names, a single asterisk is not allowed.

```
RACDCERT ID(WEBSRV) ADDRING(WEBSERVER)
```

- ▶ **Step 6.** Connect the certificate to the keyring.

Now we can create the connection between the digital certificate and the keyring with the RACDCERT CONNECT command and associate it with the HTTP started task user ID.

```
RACDCERT ID(WEBSRV) CONNECT(ID(WEBSRV) LABEL('Web Server Certificate')
                             RING(WEBSERVER) DEFAULT USAGE(PERSONAL))
```

External CA-signed server certificate gskkyman procedure

The following step-by-step example shows how to create a key database in the HFS and how to create certificate requests for external CAs for the IBM HTTP Server for OS/390. A similar procedure may be used for other server applications, including the TN3270 server and the Policy Agent.

In our test environment, we elected to have our certificate issued by the public Certificate Authority company, VeriSign. In the following discussion, we show how we created the certificate request for our server certificate, and how we received it into the key database.

It is assumed that you have set up a key database and produced a stash file for the server as discussed in 10.5, “gskkyman command use” on page 210, that you have set the correct UNIX permission bits so that the server can read the files, and that the database is named /u/takada/sslkey/server.kdb.

- ▶ **Step 1.** Create a public-private key pair and certificate request.

Figure 10-35 on page 237 shows the menu screen of the gskkyman utility. To create a public-private key pair and a certificate request, you select **3 - Create new key pair and certificate request** on this screen.

```

Key database menu

Current key database is /u/takada/sslkey/server.kdb

    1 - List/Manage keys and certificates
    2 - List/Manage request keys
    3 - Create new key pair and certificate request
    4 - Receive a certificate issued for your request
    5 - Create a self-signed certificate
    6 - Store a CA certificate
    7 - Show the default key
    8 - Import keys
    9 - Export keys
   10 - List all trusted CAs
   11 - Store encrypted database password

    0 - Exit program

Enter option number (or press ENTER to return to the parent menu): 3 1
Enter certificate request file name or press ENTER for "certreq.arm": server.arm 2
Enter a label for this key.....> ITSO Raleigh Webserver Cert 3
Select desired key size from the following options (512):
    1: 512
    2: 1024
Enter the number corresponding to the key size you want: 1 4
Enter certificate subject name fields in the following. 5
Common Name (required).....> mvs03c.itso.ral.ibm.com
  Organization (required).....> IBM Corp.
  Organization Unit (optional).....> ITSO Raleigh
  City/Locality (optional).....> Research Triangle Park
  State/Province (optional).....> North Carolina
  Country Name (required 2 characters)..> US

Please wait while key pair is created...

Your request has completed successfully, exit gskkyman? (1 = yes, 0 = no) [0]: 0

```

Figure 10-35 Create a new key pair and certificate request

- 1** Select option **3 - Create new key pair and certificate request** to create a new key pair and certificate request. If you want to create a self-signed certificate, select option **5 - Create a self-signed certificate**.
- 2** Specify a file name for the certificate request. Later you have to send the contents of this file to the CA you selected.
- 3** Enter a label related to this key and certificate.
- 4** Select the key size you desire. However, the key size depends on the location. In our test environment (ITSO Raleigh), we installed the strong crypto version of the gskkyman utility. In almost all cases, you would want to install the strong crypto version and use a key size of 1024 bits.
- 5** Enter the certificate subject name fields. Common Name should be your server's host name. If you specify another name, a user will receive the window shown in Figure 10-36 on page 238 when accessing this server via a browser.



Figure 10-36 Netscape Navigator's window checking certificate name

The only way a Web browser can check the server's identity is to compare the host name in the URL with the host name in the Common Name attribute in the certificate. If they do not match, Netscape Navigator will display the warning window shown in Figure 10-36. Whether Internet Explorer (IE) will display a warning window or simply terminate the connection depends on the release level of IE and the security level chosen.

► **Step 2.** Request the certificate from the Certificate Authority.

After step 1, you have three files in addition to the key database:

- A certificate request file (*.arm)
- A stash file (*.sth)
- A key pair file (*.rdb)

Figure 10-37 shows the contents of the certificate request file. You send this request to the Certificate Authority to be signed. We sent this certificate request to VeriSign; as shown in Figure 10-38 on page 239, you can copy and paste the contents of the request file into the VeriSign form.

```
TAKADA @ RA03:/u/takada/sslkey>ls -l
total 192
-rw-r--r--  1 TAKADA  OMVSGRP    513 Aug  6 18:56 server.arm
-rw-r--r--  1 TAKADA  OMVSGRP   65080 Aug  6 18:54 server.kdb
-rw-r--r--  1 TAKADA  OMVSGRP    5080 Aug  6 18:56 server.rdb
-rw-----  1 TAKADA  OMVSGRP    129 Aug  6 18:56 server.sth
TAKADA @ RA03:/u/takada/sslkey>cat server.arm
-----BEGIN NEW CERTIFICATE REQUEST-----
MIIBNTCB4AIBADB7MQswCQYDVQQGEwJVUzELMAkGA1UECBMxDTALBgNVBACt
BENhcnkxGDAwBgNVBAoTD01CTSBDb3Jwb3JhdG1vbJEUUBG1A1UECxMlSVRTRTYBS
YXpZ2gxIDAeBgNVBAMTF212czAzYy5pdHNvLnJhbC5pYm0uY29tMFwwDQYJKoZI
hvcNAQEBBQADSwAwSAJBaJaDyGjF0xIvb3FXm68t66tDQ+dn9B/zLthCS7dc7nor
KT6YpfjnI7duvw/zXXMrrJP99y4oLIGafHIZq1qAHO0CAwEAAMA0GCSqGSIs3
DQEBBAUAOEA70FskVCHrzZXkyoIa6NnDdrtt6CHhMKLJKtIiStFPXZVIMQxPK
1ER2vdsdzpQtIqgTromX2Jf414qm47gcWA==
-----END NEW CERTIFICATE REQUEST-----
```

Figure 10-37 The content of the certificate request file

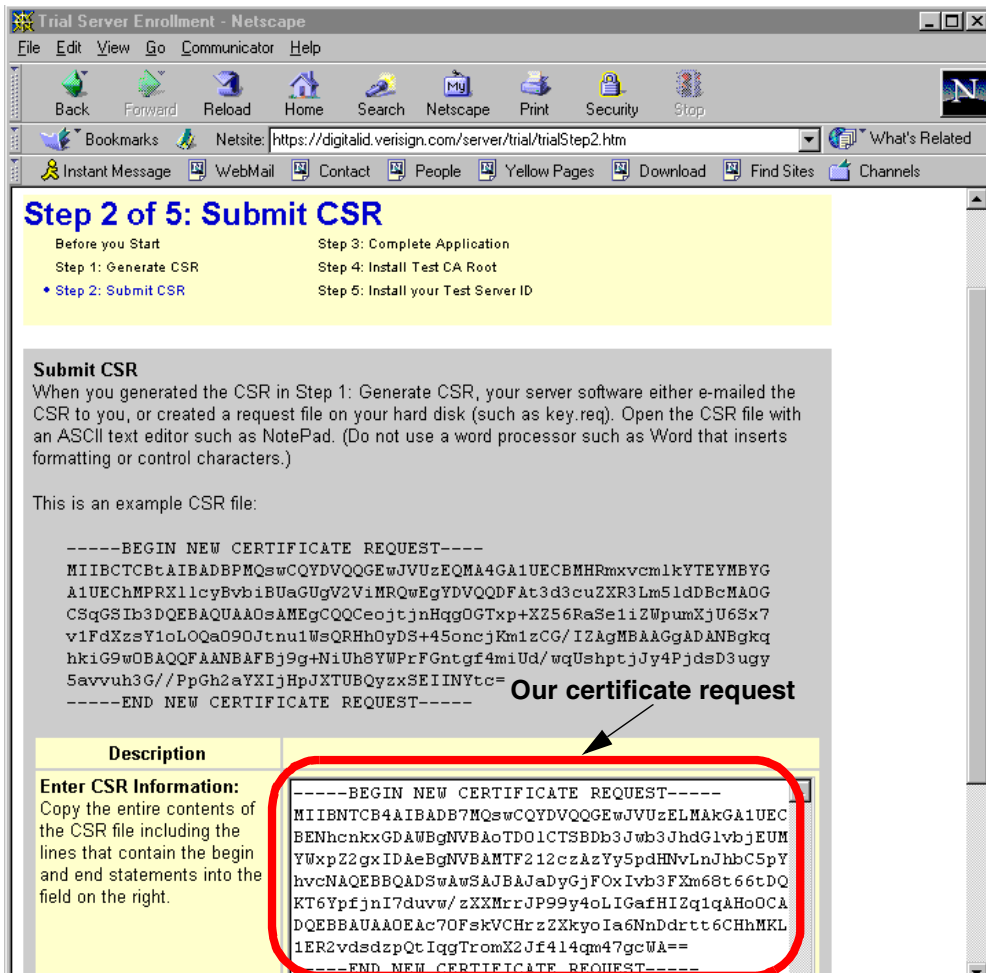


Figure 10-38 Certificate request submit form at VeriSign Web site

After you send a certificate request to an external CA, it can take several days before the request is processed and the certificate is returned. We used the trial Secure Server ID from VeriSign to test the IBM HTTP Server for z/OS SSL function in the ITSO Raleigh test environment. This provides a temporary certificate that is valid for two weeks from the date of issuance. Because the certificate is temporary, it does not require the extensive checking that a real certificate would, so we received it almost immediately after submitting the certificate request.

While you are waiting for the CA to process your certificate request, it is a good idea to exploit a trial certificate to test SSL sessions. Alternatively (or in addition), you can use the gskkyman utility to create a self-signed certificate to enable SSL sessions between clients and the server. For detailed information regarding how to make a self-signed certificate, see *IBM HTTP Server for z/OS Planning, Installing, and Using*, SC31-8690, or *Enterprise Web Serving with the Lotus Domino Go Webserver for OS/390*, SG24-2074.

► **Step 3.** Receive the certificate from the Certificate Authority.

After receiving a certificate from the CA via e-mail, copy and paste it to an HFS file. In Figure 10-39 on page 240, we created a file server.cert and put our certificate into this file.

```

File Edit Confirm Menu Utilities Compilers Test Help
-----
EDIT      /u/takada/sslkey/server.cert          Columns 00001 00072
Command ==>                               Scroll ==> CSR
***** ***** Top of Data *****
000001 -----BEGIN CERTIFICATE-----
000002 MIICJTCCAc8CEA35fK/Qad35oFktNSeSsS8wDQYJKoZIhvcNAQEEBQAwwgkxZjAU
000003 BgNVBAoTDVZlcm1TaWduLCBjbMxRzBFBgNVBAsTPnd3dy52ZXJpc21nbi5jb20v
000004 cmVwb3NpdG9yeS9UZXRNOQ1BTIE1uY29ycC4gQnkglUmVmlBMaWFiLiBMVEQuMUyW
000005 RAYDVQQLZz1Gb3IgVmVyaVNPZ24gYXV0aG9yaXplZCB0ZXN0aW5nIG9ubHkuIE5v
000006 IGFzc3VyYW5jZXMgKEMpV1MxOTk3MB4XDTk5MDgwNjAwMDAwMFoXDTk5MDgyMDIz
000007 NTk1OVowgYExCzAJBgNVBAYTA1VTMRcwFQYDVoQIEw50b3J0aCBDYXJvbGluYTEN
000008 MAsGA1UEBxQE2FyeTESMBAGA1UEChQJSUJNIEVncnAuMRQwEgYDVoQQLFAtJVFNP
000009 IFJhbGlnaDEgMB4GA1UEAxQxbXZzMDNjLm10c28ucmFsLm1ibS5jb20wXDANBgkq
000010 hkiG9w0BAQEFAANLADBIAKEA4P/8r7jWD27V1XWTP112Gg0qcakpxrTaXZ78x/Sr
000011 EMydBym0nxhrRzK21DFbpT1bM9mT+ju0av9mKiUxf19WswIDAQABMAOGCSqGS1b3
000012 DQEBAUAAOEAR20tpJvdpN4NcR6Lzx3eBGUZ4VtwtwkvKeU2AU6N9/JXOMGS2r+m
000013 IckUeu4+pRF+cHZY8uLjL1hA+c0Bux4RKA==
000014 -----END CERTIFICATE-----
***** ***** Bottom of Data *****

F1=Help      F2=Split      F3=Exit      F5=Rfind      F6=Rchange      F7=Up
F8=Down      F9=Swap       F10=Left     F11=Right     F12=Cancel .

```

Figure 10-39 The content of the server certificate issued by the trusted CA

Because the gskkyman utility accepts only HFS files, you have to create an HFS file for your certificate.

- ▶ **Step 4.** Import the CA-signed server certificate into the key database.

Figure 10-40 on page 241 shows **gskkyman** being used to import the certificate into your key database.


```

IBM Key Management Utility

Choose one of the following options to proceed.

    1 - Create new key database
    2 - Open key database
    3 - Change database password

    0 - Exit program

Enter your option number: 2 1
Enter key database name or press ENTER for "key.kdb": server.kdb
Enter password for the key database.....>*****

                Key database menu

Current key database is /u/takada/sslkey/server.kdb

    1 - List/Manage keys and certificates
    2 - List/Manage request keys
    3 - Create new key pair and certificate request
    4 - Receive a certificate issued for your request
    5 - Create a self-signed certificate
    6 - Store a CA certificate
    7 - Show the default key
    8 - Import keys
    9 - Export keys
   10 - List all trusted CAs
   11 - Store encrypted database password

    0 - Exit program

Enter option number (or press ENTER to return to the parent menu): 4 2
Enter certificate file name or press ENTER for "cert.arm": server.cert 3
Do you want to set the key as the default in your key database? (1 = yes, 0 = no) [1]:
1 4
Please wait while certificate is received.....

```

Figure 10-40 Store the server certificate into the key database

- 1** Open the key database. Since the `gskkyman` command was entered from the subdirectory `/u/takada/sslkey`, there was no need to specify the subdirectory in the key database name.
- 2** Select option **4 - Receive a certificate issued for your request** to store your server certificate file.
- 3** Specify your server certificate file created in Figure 10-39 on page 240.
- 4** You have to select 1. If you set another key as the default, server authentication will fail.

External CA-signed client certificate RACDCERT procedure

The procedure for a client to request a CA-signed certificate is dependent on the type of client software being used. Most SSL/TLS-enabled clients will have a method to create a file with a certificate request for submission to a CA.

A client certificate, in addition to being stored in the client's key database, must be added to RACF and associated with the appropriate RACF user ID using the RACFDCERT ID(clients-user ID) ADD.... statement. The client certificate's CA must be connected to the server's RACF keyring using the RACFDCERT ID(servers-user ID) CONNECT. The basic procedure to follow is:

- ▶ **Step 1.** Generate a certificate request at the client.

Clients have different ways to generate a certificate request from an external CA. For this example, we will be requesting a client certificate for a PCOMM (TN3270) client. Figure 10-41 shows the PCOMM window to request a certificate.

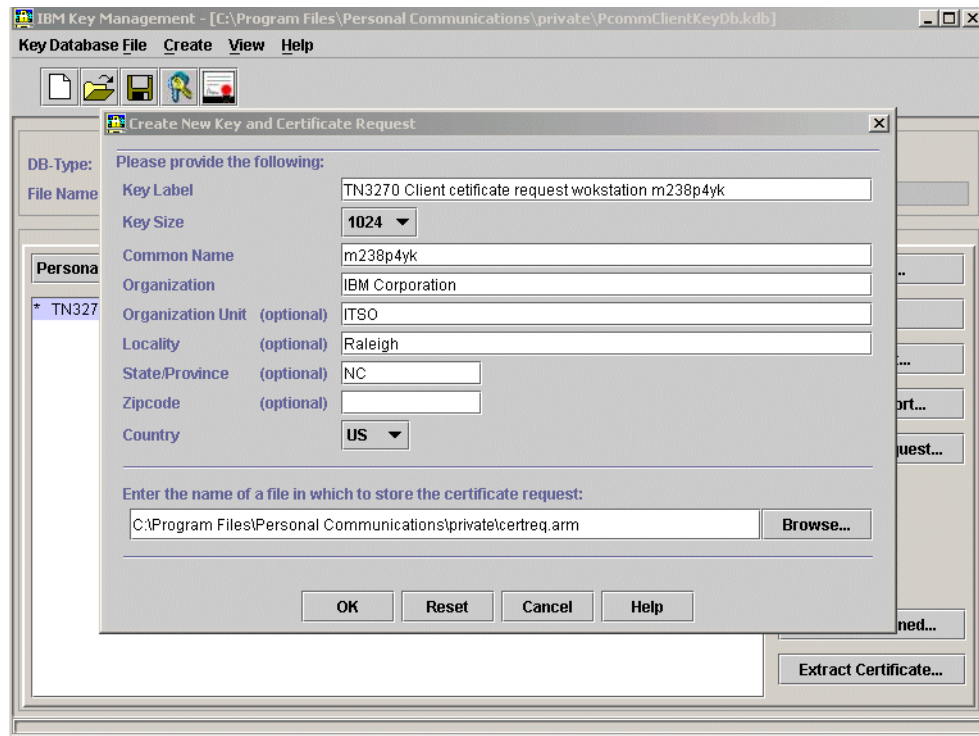


Figure 10-41 Using a client to request a certificate

See Figure 10-42 for an example of the certificate request file that is created.

```

-----BEGIN NEW CERTIFICATE REQUEST-----
MIIBqDCCARECAQAwDELMAkGA1UEBhMCVVMxMzA3bG9yYXRpb24xDTALBgNVBAsTBTEU
U08xETAPBgNVBAMTCG0yMzhwNH1rMIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKB
gQC81RfEw/spXrdZY/eSES6kFkrI+Bv011VhYQ+X/+1gsA/Bbb85e75hsPAHU/+q
xeDC2JDqJrjPIChbwxBOmRofxwYhSpu51grQJIYYMehbw1mz9BvF3V+I8SV2fp+A
uPXtjw17cTC1tNO+mbBn1xgYVDaygOgkh8Xh1M4QMderIwIDAQBoAAwDQYJKoZI
hvcNAQEEBQADgYEAHdJbb3R3i7a2WJgQKn1+TdbeJxX9D8bdufXfzwCRcLqBPni
kVeh6Hg5z+UeLX70+Cr3TsPmYJHAXZYQCNCATcsHIRj1p5XC50VDrckEG/RpVLvf0
36Y2fYOT4f86sOy8L2RwhRSm3V2mC5vG9Jj1B1MS2hkQ13ZWFkYrFMvvczo=
-----END NEW CERTIFICATE REQUEST-----

```

Figure 10-42 Certificate request generated by client for external CA

- ▶ **Step 2.** Send the certificate request to the Certificate Authority.

E-mail the certificate request output from the client, either by using cut and paste, or as an attachment. The CA may take a number of days to generate and send the certificate and private key back to you.

- ▶ **Step 3.** Receive the certificate from the CA.

Depending on the CA, you may have to go to a secure Web site to download your client certificate and key or they may send it to you in a secure e-mail. Whatever method is chosen, you must end up with a file, probably in PKCS12 format, which contains both a digital certificate and a private key. This file will be password protected.

- ▶ **Step 4.** Import the client certificate (and private key) into the client key database.

In step 3, the certificate and key were received and detached into a workstation file. That certificate and key must now be imported into your client's key database. As an example, we show how to import the client certificate and key into the PCOMM key database.

First, start the Certificate Management utility. Click **Start -> Programs -> IBM-Personal-Comm -> Utilities -> Certificate Management**.

Open the PCOMM key database and give the password (the default installation password is "PCOMM").

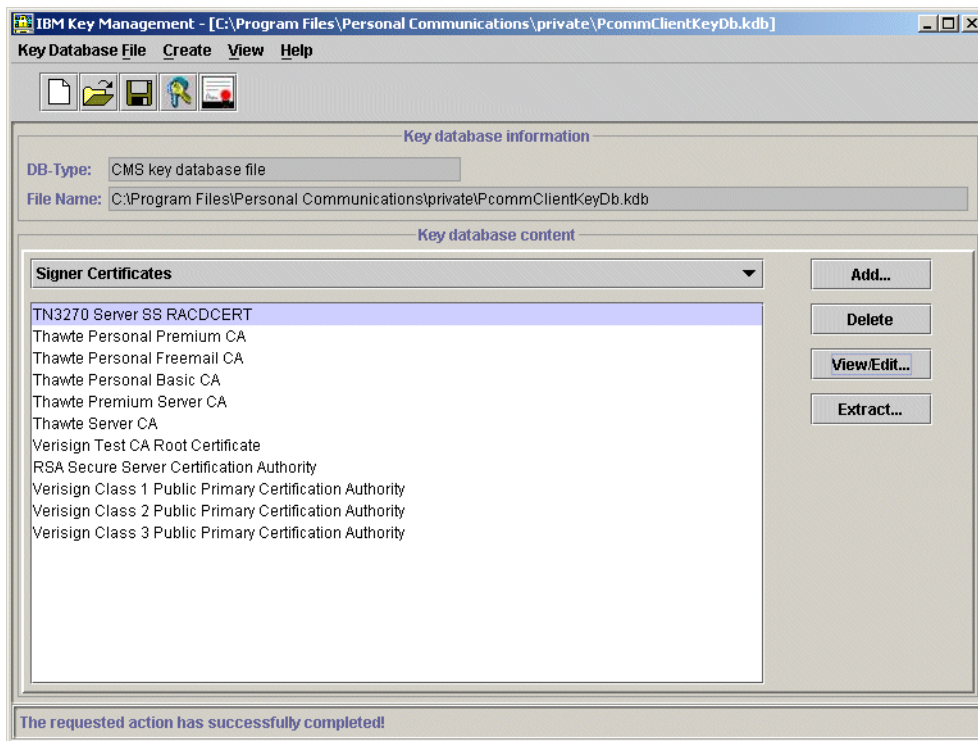


Figure 10-43 PCOMM certificate management window after key database is opened

Figure 10-43 shows the window that is displayed after the PCOMM key database is opened. The highlighted certificate is the server certificate that we have already assumed to have been set up.

Just below the heading “Key database content” there is a drop-down box containing the words “Signer Certificates”, so the certificates listed are the CA certificates. Click the drop-down box and select **Personal Certificates**. You will then be presented with a list of personal certificates. If you have never imported any, the list will be blank. In any case, then click the **Import** button and select the certificate file you exported in step 1, and click **OK**. You will be asked for a password, which will have been provided by your CA.

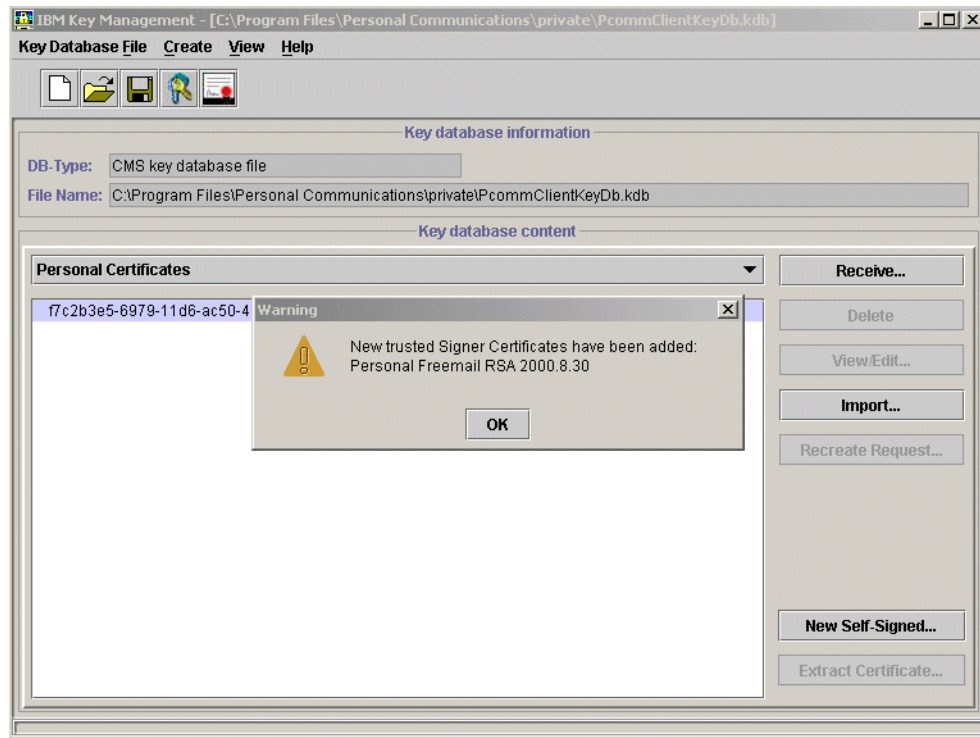


Figure 10-44 PCOMM certificate dialog after importing a client certificate and key

Figure 10-44 shows the window after you have imported the key/certificate into the key database. As the file received from the CA not only contained the client certificate, but also the signer’s certificate, the import function added two certificates to the client key database, one in the Personal Certificates section and one in the Signer Certificates section as indicated in the informational window.

► **Step 5.** Export the client certificate to a workstation file.

This step is not shown for brevity. It is a matter of highlighting the client certificate in the Personal Certificates section and clicking the **Export/Import...** button, then following the instructions to create a Base64-encoded ASCII certificate file. Note that the file received from the CA should not be sent to the server, since it contains the client’s private key. That is why it is first imported into the client’s key database, then just the certificate is exported, not the private key.

Note: This same step might need to be followed for the CA certificate as well. If the CA that signed your client certificate does not have a CA certificate at the server’s key database, then the server has no way of validating the client certificate. You will need to export the CA certificate, add it into the RACF database and connect it as a CA certificate into the server’s keyring. Use the RACDCERT CERTAUTH LIST command to see all Certificate Authority certificates in your system.

- ▶ **Step 6.** FTP the client (and CA certificate if needed as per the note in step 4) exported in step 5 to the z/OS system as MVS data set(s).

This step does not need to be shown. Ensure the transfer is a text-type transfer to enable ASCII/EBCDIC translation.

Important: The RACF RACDCERT command requires a certificate file that it will be importing to be in variable blocked format. To do this from a workstation FTP client, use the **quote site recfm=vb** command or if using a GUI FTP client, consult the help files.

- ▶ **Step 7.** Add the client certificate into the RACF database as a TRUSTED certificate for the RACF ID that you want to associate the certificate with (the user of the workstation).

In our case, the client certificate was issued for user ID “FOCAS”, and the FTP in step 6 transferred the client certificate to an MVS data set named FOCAS.RACDCERT.THAWTE.CLIENT.CERT.

```
//CLIENT EXEC PGM=IKJEFT01,DYNAMNBR=30,REGION=4096K
/*
/* Import the Client certificate issued by Thawte, and set the
/* TRUST flag on.
/*
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *
RACDCERT ID(FOCAS) ADD('FOCAS.RACDCERT.THAWTE.CLIENT.CERT') -
WITHLABEL('Thawte Certificate for FOCAS') TRUST
/*
```

Figure 10-45 Batch job to import client certificate and associate with a RACF user ID

Figure 10-45 shows the RACDCERT ADD command being used to add the client certificate issued by the CA into the RACF database. The TRUST flag is set on, and the ID(FOCAS) parameter associated this certificate with the user ID FOCAS. This certificate does not need to be added to the server’s keyring.

- ▶ **Step 8.** Connect the CA certificate of the client certificate’s issuer into the RACF database and connect it to the server’s keyring.

If the CA certificate is already in the RACF database, the ADD step can be skipped, but you must still connect the CA certificate to the server’s keyring with USAGE(CERTAUTH), which sets the TRUST status on for use in the server’s keyring.

```

//CACERT EXEC PGM=IKJEFT01,DYNAMNBR=30,REGION=4096K
/**
/** Import the CA Certificate that signed the client certificate
/** into the RACF database. This certificate was FTP'd from the
/** TN3270 client PCOMM key database from the 'Servers Certifictes'
/** section.If the CA certificate is already in the RACF database
/** the 'ADD' step can be skipped. You must still add the CA
/** certificate to the servers keyring with USAGE(CERTAUTH)
/**
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *
RACDCERT CERTAUTH ADD('FOCAS.RACDCERT.FREEMAIL.RSA.CERT') -
WITHLABEL('Thawte Freemail RSA 2000') notrust
RACDCERT ID(STC) CONNECT(CERTAUTH -
LABEL('Thawte Freemail RSA 2000') -
RING(TN327ORing) -
USAGE(CERTAUTH))
/*

```

Figure 10-46 Batch job to import CA certificate and add to server's RACF keyring as a CA

Figure 10-46 shows the RACDCERT ADD command being used to add the CA certificate into the RACF database. In this example, we added the certificate as a CA certificate (CERTAUTH on the ADD) but set the NOTRUST flag. This is overridden in any case when you connect the CA certificate to the server's keyring as USAGE(CERTAUTH). This was just an example to show how the TRUST status of a certificate can be overridden for a particular server's use by adding that certificate to the server's keyring as a CA certificate by specifying USAGE(CERTAUTH).

Table 10-2 Example certificate locations

Certificate	z/OS RACF Database	PCOMM Client Key Database
Client Certificate	ADDED with ID(FOCAS) status TRUST If self-signed, must also CONNECT to server's keyring as USAGE(CERTAUTH)	In the Personal Certificates section as "default" certificate
Client Certificate's CA certificate	ADDED with CERTAUTH (trust status immaterial) CONNECT to server's keyring as USAGE(CERTAUTH)	In the Signer Certificates section marked as "trusted root"
Server Certificate	ADDED with ID(STC) CONNECT to server's keyring as USAGE(PERSONAL) and DEFAULT	Not needed unless the server certificate is a self-signed in which case it is added to the Signer Certificates section as a "trusted root".
Server Certificates CA certificate (non-self-signed server certificate only)	ADDED with CERTAUTH (trust status immaterial) CONNECT to server's keyring as USAGE(CERTAUTH)	In the Signer Certificates section as a "trusted root".

Table 10-2 shows the various locations of the certificates for client authentication to happen. The assumption is that user FOCAS is the client user and STC is the RACF user ID associated with the server. If the server uses a self-signed certificate, ignore the Server Certificate CA row. If the client uses a self-signed certificate, ignore the Client Certificate CA row.

Note: The RACF database comes pre-configured with CA certificates from some major Certificate Authorities. These companies are Thawte and VeriSign. If you delete these CA certificates from the RACDCERT database, they will be reinstated automatically after an IPL.

Once these steps are followed, the connection can be established using SSL/TLS.

External CA-signed client certificate gskkyman procedure

The basic procedure to follow for **gskkyman** is:

- ▶ **Steps 1 through 4** are exactly the same as that outlined in “External CA-signed client certificate RACDCERT procedure” on page 241.
 - a. Generate a certificate request at the client
 - b. Send the certificate request to the Certificate Authority
 - c. Receive the client certificate from the CA
 - d. Import the certificate into the client’s key database
- ▶ **Step 5.** When you use **gskkyman**, you do not store the client’s certificate in the server’s keyring. However, you need to ensure that the CA that issued the client certificate has a CA certificate in the server’s keyring. This is needed so that the server can validate the client’s certificate when the client presents it during the SSL handshake. The CA certificate should already be in the key database as a CA. The **gskkyman** database comes pre-configured with CA certificates from some major Certificate Authorities such as Thawte and VeriSign. If your client certificate was issued by some other CA that does not have a CA certificate in the server’s HFS key database, you should export it from the client, where it would have been also added when you imported your certificate from the CA, and then add it to the **gskkyman** database by selecting option **6 - Store a CA certificate**.

10.10 Certificate locations example

This section briefly summarizes the locations of certificates using each of the utilities.

10.10.1 RACF certificates

The following example may help you visualize where client and server certificates must be located in order for SSL/TLS to function correctly. Both user1 and user2 are clients of the FTP and TN3270 servers on z/OS. Both clients use a workstation-based TN3270 client. user1 uses a z/OS-based FTP client and user2 has a workstation FTP client.

Table 10-3 Example list of certificates for RACF database

Owner	Certificate Number	Signer	Description	Trusted?
ID(user1)	1	Self	FTP Client Certificate	Yes
ID(user1)	2	VeriSign	TN3270 Client Certificate	Yes
ID(user2)	3	VeriSign	FTP Client Certificate	Yes
ID(user2)	4	Self	TN3270 Client Certificate	Yes
ID(FTP)	5	VeriSign	FTP Server Certificate	Don't care
ID(TN3270)	6	Self	TN3270 Server Certificate	Don't care

Owner	Certificate Number	Signer	Description	Trusted?
CERTAUTH	7	Self	VeriSign CA Certificate	Don't care

Certificate table description

In Table 10-3 on page 247, the 'Owner' column identifies the certificate owner. This is the parameter used on the RACDCERT command when you add, delete or change this certificate entry.

user1 is an FTP and TN3270 client's RACF user ID. His FTP client is on z/OS and his TN3270 client is on a workstation. The FTP client has a self-signed certificate and the TN3270 workstation client has a certificate issued by a well-known CA, VeriSign.

user2 is an FTP and TN3270 client's RACF user ID. His FTP client and TN3270 client are on a workstation. The TN3270 client has a self-signed certificate and the FTP client has a certificate issued by a well-known CA, VeriSign.

FTP is the RACF user ID that the TLS-enabled FTP server on z/OS runs under. The FTP server has a server certificate issued by a well-known CA, VeriSign.

TN3270 is the RACF user ID that the SSL-enabled TN3270 server on z/OS runs under. The TN3270 server has a self-signed server certificate.

CERTAUTH is that part of the RACF database reserved for CA certificates. The VeriSign certificate is shown as being self-signed, although it could also be signed by a higher-level authority. For the purposes of this discussion, that is immaterial.

Keyrings needed for example

If we discuss each user in turn, we can see where the digital certificates for the clients and the servers need to be located. We are assuming both the TN3270 and FTP servers are configured for client authentication (server authentication is mandatory).

user1

Table 10-4 RACF keyring connections needed for user1

Keyring name	Keyring association	Certificate number	Certificate association	Default	Usage()
FTPClientRing	ID(user1)	1 ¹	ID(user1)	Yes	PERSONAL
FTPClientRing	ID(user1)	7 ²	CERTAUTH	No	CERTAUTH

user1 is an FTP and TN3270 client user. The FTP client is on z/OS and will be communicating with the z/OS-based TLS-enabled FTP server. Therefore the client needs a keyring to store the certificates that will be used in the SSL exchange. Table 10-4 shows the RACF keyring with example name "FTPClientRing" set up for the user. There are two certificates in the keyring:

- ¹ The client certificate (certificate number 1 in Table 10-3 on page 247) is needed by the client to pass to the FTP server when requested as part of client authentication. The client certificate must be in "TRUSTED" status and must be set to the DEFAULT certificate in the keyring. This is how the FTP client knows which certificate to pass in the TLS exchange.
- ² The server that is being connected to has a server certificate (certificate number 5 in Table 10-3 on page 247) signed by an external CA, VeriSign (certificate number 7 in Table 10-3 on page 247). The CA certificate is needed by the client to validate the server certificate passed by the FTP server during the TLS exchange.

user1's TN3270 client is not on z/OS so does not require a RACF keyring. However, the TN3270 client on the workstation will require its key database to contain the client certificate and a CA certificate to verify the TN3270 server (since the TN3270 server certificate is self-signed, the server certificate itself *is* the CA certificate).

user2

user2 is an FTP and TN3270 client user, with both clients on a workstation. Therefore, user2 does not require a RACF keyring. However, the workstation key databases of both clients need the following:

- ▶ The TN3270 client on the workstation will require its key database to contain user2's TN3270 client certificate and a CA certificate to verify the TN3270 server (since the TN3270 server certificate is self-signed, the server certificate itself *is* the CA certificate).
- ▶ The FTP client on the workstation will require its key database to contain the user2's FTP client certificate and a CA certificate to verify the FTP server (since the FTP server certificate was signed by VeriSign, you should ensure that the VeriSign CA certificate is present and set as "TRUSTED").

FTP

Table 10-5 RACF keyring connections needed for user FTP

Keyring name	Keyring association	Certificate number	Certificate association	Default	Usage()
FTPServerRing	ID(FTP)	5 1	ID(FTP)	Yes	PERSONAL
FTPServerRing	ID(FTP)	1 2	ID(user1)	No	CERTAUTH
FTPServerRing	ID(FTP)	7 3	CERTAUTH	No	CERTAUTH

FTP is the RACF user ID that the z/OS-based FTP server runs under. Any TLS-enabled server requires a keyring to refer to certificates that it will be using in the TLS exchange. A certificate is required for the server, and a certificate is required for every different CA that has issued the FTP client certificates. Table 10-5 shows the RACF keyring with example name "FTPServerRing". There are three certificates in the keyring:

- 1** When the server sends its certificate to the client at the beginning of the TLS handshake, it looks in the keyring (whose name is in the server's configuration file) for a DEFAULT certificate. That is the server's certificate that is passed to the client (certificate number 5 in Table 10-3 on page 247).
- 2** This is the CA certificate to validate user1. Since user1 had a self-signed client certificate for FTP, the client certificate (certificate number 1 in Table 10-3 on page 247) is being used here in the context of a CA certificate, that is, it will be used to validate the client certificate when it is passed to the server.
- 3** This is the CA certificate to validate user2. Since user2 had a client certificate for FTP issued by an external company, VeriSign, this is the VeriSign CA certificate (certificate number 7 in Table 10-3 on page 247).

TN3270

Table 10-6 List of RACF keyrings needed for example

Keyring name	Keyring association	Certificate number	Certificate association	Default	Usage()
TN3270ServerRing	ID(TN3270)	6 1	ID(TN3270)	Yes	PERSONAL
TN3270ServerRing	ID(TN3270)	4 2	ID(user2)	No	CERTAUTH

Keyring name	Keyring association	Certificate number	Certificate association	Default	Usage()
TN3270ServerRing	ID(TN3270)	7 3	CERTAUTH	No	CERTAUTH

TN3270 is the RACF user ID that the z/OS based TN3270 server runs under. Any SSL-enabled server requires a keyring to refer to certificates that it will be using in the SSL exchange. A certificate is required for the server, and a certificate is required for every different CA that has issued the TN3270 client certificates. Table 10-6 on page 249 shows the RACF keyring with example name 'TN3270ServerRing'. There are three certificates in the keyring:

- 1** When the server sends its certificate to the client at the beginning of the SSL handshake, it looks in the keyring (whose name is in the server's configuration file) for a 'DEFAULT' certificate. That is the servers certificate that is passed to the client. (certificate number 6 in Table 10-3 on page 247).
- 2** This is the CA certificate to validate user2. As user2 had a self-signed client certificate for TN3270, the client certificate (certificate number 4 in Table 10-3 on page 247) is being used here in the context of a CA certificate, that is, it will be used to validate the client certificate when it is passed to the server.
- 3** This is the CA certificate to validate user1. Since user1 had a client certificate issued by an external company, VeriSign, this is the VeriSign CA certificate (certificate number 7 in Table 10-3 on page 247).

10.10.2 gskkyman HFS certificates

The following example may help you visualize where client and server certificates must be located in order for SSL/TLS to function correctly. Both user1 and user2 are clients of the FTP and TN3270 servers on z/OS. Both clients use a workstation-based TN3270 client. user1 uses a z/OS-based FTP client and user2 has a workstation FTP client.

Table 10-7 Example list of certificates for examples to follow

Certificate Subject	Certificate Number	Signer	Description	Trusted?
user1	1	Self	FTP Client Certificate	Yes
user1	2	VeriSign	TN3270 Client Certificate	Yes
user2	3	VeriSign	FTP Client Certificate	Yes
user2	4	Self	TN3270 Client Certificate	Yes
FTP	5	VeriSign	FTP Server Certificate	No
TN3270	6	Self	TN3270 Server Certificate	Yes
VERISIGN	7	Self	VeriSign CA Certificate	Yes

Certificate table description

Table 10-7 shows a list of certificates we will be using to set up some different key databases. A key database will be set up for the RACF user IDs user1, FTP, and TN3270.

user1 is an FTP and TN3270 client's RACF user ID. His FTP client is on z/OS and his TN3270 client is on a workstation. The FTP client has a self-signed certificate and the TN3270 workstation client has a certificate issued by a well-known CA, VeriSign.

user2 is an FTP and TN3270 client's RACF user ID. His FTP client and TN3270 client are on a workstation. The TN3270 client has a self-signed certificate and the FTP client has a certificate issued by a well-known CA, VeriSign.

FTP is the RACF user ID that the TLS-enabled FTP server on z/OS runs under. The FTP server has a server certificate by a well-known CA, VeriSign.

TN3270 is the RACF user ID that the SSL-enabled TN3270 server on z/OS runs under. The TN3270 server has a self-signed server certificate.

VERISIGN is an external CA certificate. The VeriSign certificate is shown as being self-signed, although it could also be signed by a higher-level authority. As long as the "trusted root" status is set, that is OK.

HFS key databases needed for example

If we discuss each user in turn, we can see where the digital certificates for the clients and the servers need to be located. We are assuming both the TN3270 and FTP servers are configured for client authentication (server authentication is mandatory).

user1

Table 10-8 Key database needed for user1 to use FTP client on z/OS

Certificate Subject	Certificate Number	Signer	Description	Default?
user1	1 ¹	Self	FTP Client Certificate	Yes
VERISIGN	7 ²	Self	VeriSign CA Certificate	No

user1 is an FTP and TN3270 client user. The FTP client is on z/OS and will be communicating with the z/OS-based TLS-enabled FTP server. Therefore the client needs a keyring to store the certificates that will be used in the SSL exchange. Table 10-8 shows the contents of the HFS key database set up for user1. There are two certificates in the key database:

- ¹ The client certificate (certificate number 1 in Table 10-7 on page 250) is needed by the client to pass to the FTP server when requested as part of client authentication. The client certificate must be in "TRUSTED" status and must be set to the DEFAULT certificate in the key database. This is how the FTP client knows which certificate to pass in the TLS exchange.
- ² The server that is being connected to has a server certificate (certificate number 5 in Table 10-7 on page 250) signed by an external CA, VeriSign (certificate number 7 in Table 10-7 on page 250). The CA certificate is needed by the client to validate the server certificate passed by the FTP server during the TLS exchange.

user1's TN3270 client is not on z/OS so does not require an HFS key database. However, the TN3270 client on the workstation will require its key database to contain user1's TN3270 client certificate and a CA certificate to verify the TN3270 server (since the TN3270 server certificate is self-signed, the server certificate itself *is* the CA certificate).

user2

user2 is an FTP and TN3270 client user, with both clients on a workstation. Therefore, user2 does not require an HFS key database. However, the workstation key databases of the FTP and TN3270 clients need the following:

- ▶ The TN3270 client on the workstation will require its key database to contain user2's TN3270 client certificate and a CA certificate to verify the TN3270 server (since the TN3270 server certificate is self-signed, the server certificate itself *is* the CA certificate).

- ▶ The FTP client on the workstation will require its key database to contain the user2's FTP client certificate and a CA certificate to verify the FTP server (since the FTP server certificate was signed by VeriSign, you should ensure that the VeriSign CA certificate is present and set as "TRUSTED").

FTP

Table 10-9 Key database needed for FTP server on z/OS

Certificate Subject	Certificate Number	Signer	Description	Default?
FTP	5 1	VeriSign	FTP Server Certificate	Yes
user1	1 2	Self	FTP Client Certificate	No
VERISIGN	7 3	Self	VeriSign CA Certificate	No

FTP is the RACF user ID that the z/OS based FTP server runs under. Any TLS-enabled server requires a key database to refer to certificates that it will be using in the TLS exchange. A certificate is required for the server, and a certificate is required for every different CA that has issued the FTP client certificates. Table 10-9 shows the contents of the HFS key database set up for user FTP. There are three certificates in the keyring:

- 1** When the server send its certificate to the client at the beginning of the TLS handshake, it looks in the key database (whose name is in the server's configuration file) for a DEFAULT certificate. That is the servers certificate that is passed to the client (certificate number 5 in Table 10-7 on page 250).
- 2** This is the CA certificate to validate user1. Since user1 had a self-signed client certificate for FTP, the client certificate (certificate number 1 in Table 10-7 on page 250) is being used here in the context of a CA certificate, that is, it will be used to validate the client certificate when it is passed to the server.
- 3** This is the CA certificate to validate user2. Since user2 had a client certificate for FTP issued by an external company, VeriSign, this is the VeriSign CA certificate (certificate number 7 in Table 10-7 on page 250).

TN3270

Table 10-10 Example list of certificates for examples to follow

Certificate Subject	Certificate Number	Signer	Description	Default?
TN3270	6 1	Self	TN3270 Server Certificate	Yes
user2	4 2	Self	TN3270 Client Certificate	No
VERISIGN	7 3	Self	VeriSign CA Certificate	No

TN3270 is the RACF user ID that the z/OS based TN3270 server runs under. Any SSL-enabled server requires a key database to refer to certificates that it will be using in the SSL exchange. A certificate is required for the server, and a certificate is required for every different CA that has issued the TN3270 client certificates. Table 10-10 shows the contents of the HFS key database set up for user TN3270. There are three certificates in the keyring:

- 1** When the server send its certificate to the client at the beginning of the SSL handshake, it looks in the key database (whose name is in the server's configuration file) for a DEFAULT certificate. That is the server's certificate that is passed to the client (certificate number 6 in Table 10-7 on page 250).
- 2** This is the CA certificate to validate user2. Since user2 had a self-signed client certificate for TN3270, the client certificate (certificate number 4 in Table 10-7 on page 250) is being

used here in the context of a CA certificate, that is, it will be used to validate the client certificate when it is passed to the server.

- 3 This is the CA certificate to validate user1. Since user1 had a client certificate issued by an external company, VeriSign, this is the VeriSign CA certificate (certificate number 7 in Table 10-7 on page 250).



File-related applications

In this chapter we talk about the security considerations for file transfer applications. These are, specifically, TFTP (Trivial File Transfer Program) and FTP (File Transfer Program), which are covered in the following sections:

- ▶ 11.1, “z/OS FTP server” on page 256
- ▶ 11.2, “z/OS TFTP server” on page 287

11.1 z/OS FTP server

File Transfer Protocol (FTP) is *primarily* used to transfer files between TCP/IP hosts. We say primarily because with z/OS, FTP can also be used to interact with JES and DB2. This redbook is concerned with the security aspects of FTP. These security concerns cover the use of RACF and protecting the data flows in an FTP session using TLS or Kerberos. For information on configuring the FTP server, see *Communications Server for z/OS V1R2 TCP/IP Implementation Guide Volume 2: UNIX Applications*, SG24-5228.

11.1.1 FTP using Transport Layer Security(TLS)

TLS, a protocol developed by Netscape, has evolved from a string of Secure Sockets Layer (SSL) versions. SSL is a way to make a secure connection between an FTP client and FTP server that support SSL. When an SSL connection is established, all data passing from one side to the other will be encrypted and will be able to be decrypted only by the parties involved in the transfer of data. This protocol uses encryption to provide confidentiality and to authenticate between applications. TLS is based on RFC 2246, which is the standard version of SSL. Securing FTP with TLS is a level of security support that is present in both the z/OS FTP client and the FTP server.

Some of the terms you come across when dealing with TLS/SSL are:

- ▶ **Digital Certificate:** The certificate holds the identification information of the client or server. It is used during connection negotiations to identify the parties involved. In some cases, the client's certificate must be validated by the server in order to open an SSL connection.
- ▶ **Session Key:** The session key is what both the client and the server use to encrypt data. It is created by the client.
- ▶ **Public Key:** The public key is the device with which the client encrypts a session key. It does not exist as a file, but is a by-product of the creation of a certificate and private key. Data encrypted with a public key can only be decrypted by the private key that made it.
- ▶ **Private Key:** The private key decrypts the client's session key that is encrypted by a public key. The private key file has the .key ending. Private keys should *never* be distributed to anyone.
- ▶ **KeyRing:** The keyring contains the server certificate and keys to be used by the FTP server and any certificates required to do client authentication checks.
- ▶ **Stash File:** Contains the keyring password file for an associated keyring file. This password file contains the encrypted password. The stash files extension is .sth, and has to be located in the same /HFS directory and the keyring file. If RACF is used, then the stash file is not required.

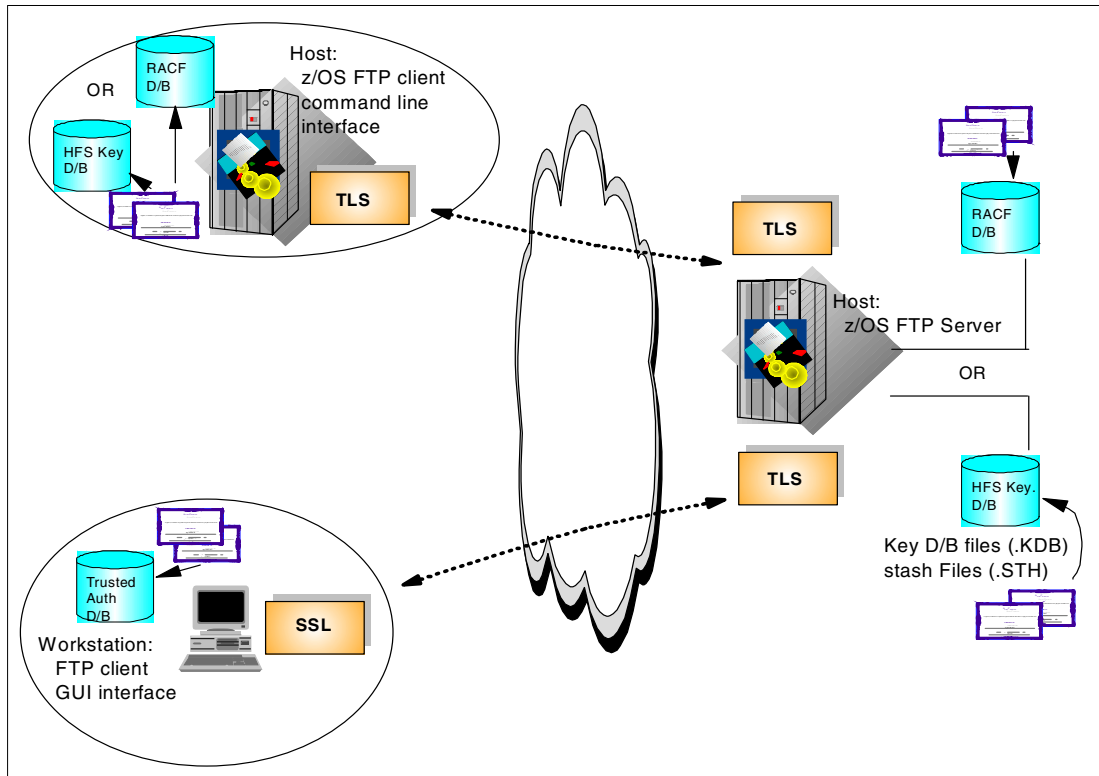


Figure 11-1 TLS/SSL security

TLS/SSL works as follows:

- ▶ The client makes the initial connection with the server and requests that an SSL connection be set up. As shown in Figure 11-1, a client can be either PC based, UNIX based, or z/OS based.
- ▶ The client compares the certificate from the server to a trusted authorities database. If the Server Certificate Authority (CA) is listed there, it means the client trusts the issuer of the server certificate and will continue the session setup.
- ▶ If the server requires a client certificate, further negotiation is done and the server and client check their respective databases for valid certificates.
- ▶ If these certificates are found, then a secure session setup is done.
- ▶ From a z/OS perspective, the FTP.DATA file specifies the level of authentication required for an FTP connection.
- ▶ In z/OS the FTP client has start options that control the client's behavior.
- ▶ The FTP client negotiates the level of security based on its parameters and the capabilities of the server. Likewise, the FTP server is configured for a certain level of support and must react to the client commands as appropriate.
- ▶ Protected port 990 - assumes TLS protection for both client and server.

The following commands are available in the server for manipulating the security environment.

AUTH	Authentication/Security Mechanism
PBSZ	Protection Buffer Size
PROT	Data Connection Protection Level

The client commands for manipulating the security environment are:

CLEAR	Sets the protection level for data transfers to CLEAR
PRIVATE	Sets the protection level for data transfers to PRIVATE
PROTECT parm	Sets the protection level for data transfers to the value of parm - either CLEAR or PRIVATE

Note: Be aware that TLS does not allow subcommands that change the protection level of the control connection. The control connection is always protected (private).

Different levels of authentication can take place when securing your FTP sessions. This would depend on the security system you wish to deploy to administer your security. The two available methods of administering security are gskkyman and RACF security server. The level of security that each of them provides are as follows:

- ▶ Using gskkyman
 - Server Authentication only
 - Client Authentication Level 1
- ▶ Using RACF
 - Server Authentication only
 - Client Authentication Level 1
 - Client Authentication Level 2
 - Client Authentication Level 3

Client Authentication Level 1 means that the client passes a certificate to the server as part of the SSL handshake. To pass authentication, the Certificate Authority (CA) that signed the client certificate must be considered trusted by the server. That is, the certificate for the CA must be in the keyring used by the server and designated as trusted.

Client Authentication Level 2 means that the client certificate has to be registered with RACF (or any other SAF-compliant security product) and mapped to a user ID. This is in addition to the checking done as described in Client Authentication Level 1. The client certificate received during the SSL handshake is used to query RACF to verify that the certificate maps to a user ID known to the system prior to the session setup. The user thus has to have a valid RACF user ID on the server host.

Client Authentication Level 3 means that in addition to level 1 and level 2 support, access is also restricted to the server based on the user ID returned from RACF. In some cases a certificate may be valid and mapped to a user ID but should be valid for only one of several servers. The third level of control uses the SERVAUTH RACF class to restrict access to the server based on client user ID.

To select the level of authentication within the FTP server, the FTP.DATA file has to contain one of the following:

- ▶ Client Authentication Level 1 `SECURE_LOGIN REQUIRED`
- ▶ Client Authentication Level 2 `SECURE_LOGIN VERIFY USER`
- ▶ Client Authentication Level 3 `SECURE_LOGIN VERIFY USER`
- ▶ RACF comes into play and differentiates between Level 2 and Level 3 authentication

The z/OS FTP server can be configured to run in two modes:

- ▶ Conditional mode uses a single port for both TLS and non-TLS FTP control connections.
- ▶ Unconditional mode uses a separate port for all TLS traffic. Port 990 is the port designated for control connections for unconditional TLS mode.

Figure 11-2 shows the AUTH command flows when setting up a TLS session using port 21 (conditional mode). The PBSZ and PROT commands are issued because the statement SECURE_DATACONN PRIVATE was coded in the FTP.DATA file.

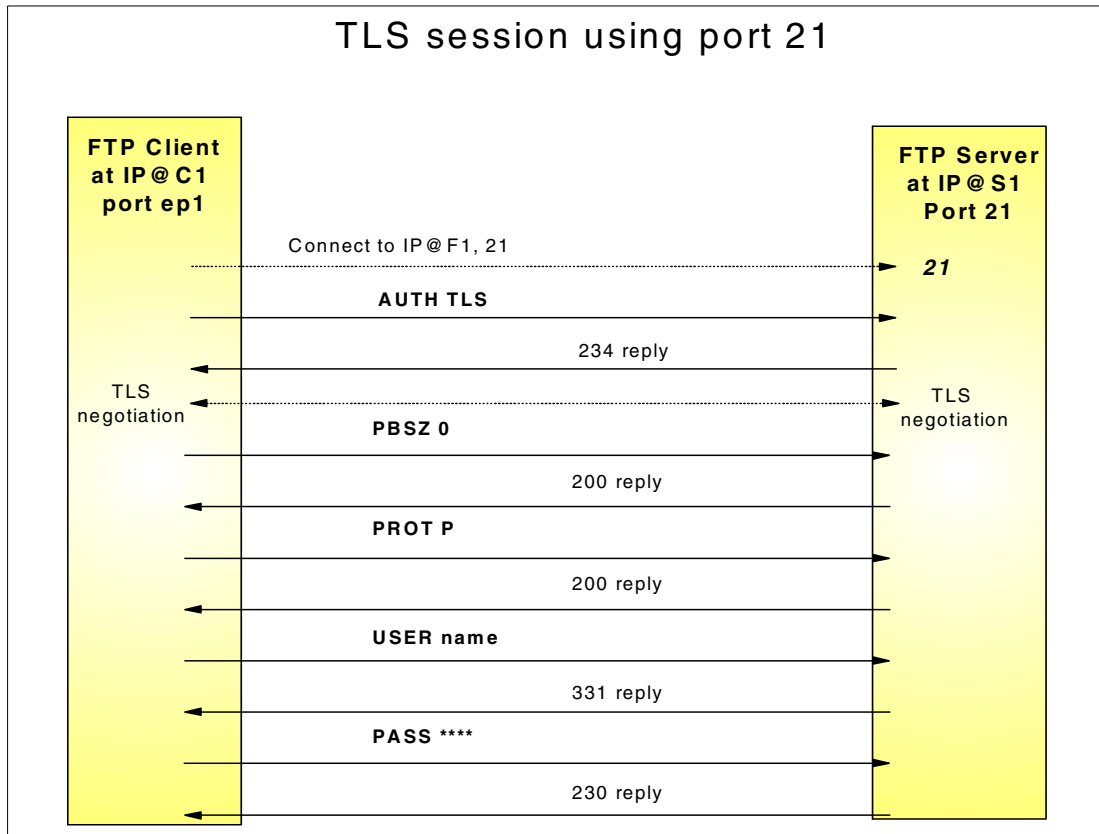


Figure 11-2 Port 21 session flows

In unconditional mode, no AUTH command flows because both client and server know that port 990 is the secure (protected) port. TLS security is assumed by both, so that immediate handshake occurs after connection. If the FTP server uses port 990 for its control connection, both the FTP client and server treat the control connection and the data connections as if they are protected by TLS. Port 990 is, however, not required for a TLS connection to be established. It is only that when port 990 is used, TLS is assumed. Port 990 can be specified for the server as follows:

In the ETC.SERVICES data set you could specify:

```
#
#      FTP TLS control connection
#
ftp          990/tcp
```

Figure 11-3 ETC.SERVICES for FTP's port 990

In the procedure used to start the FTP server, specify the start parameter (option) PORT 990.

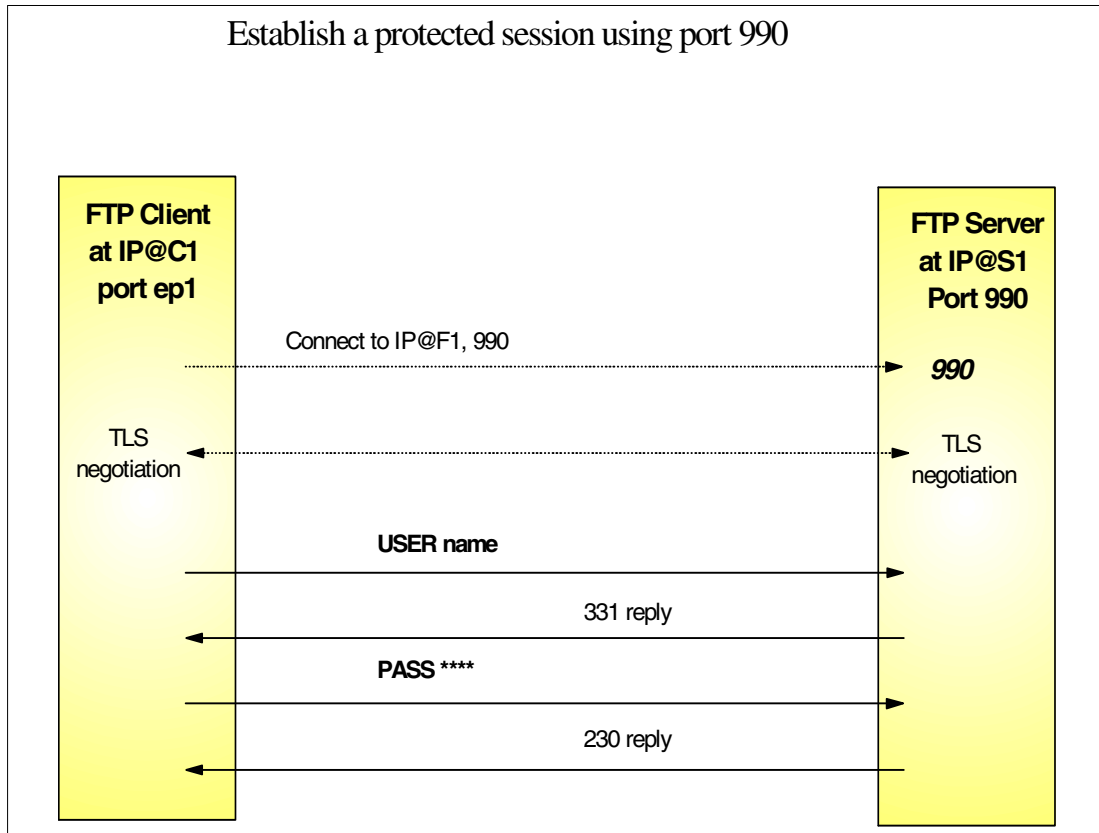


Figure 11-4 Port 990 session flows

11.1.2 TLS/SSL scenarios

For the scenarios that follow, we used self-signed certificates only. For a more thorough discussion of server and client certificates, see 2.2.4, “Digital certificates” on page 13 and Chapter 10, “Certificate management in z/OS” on page 203. The latter shows the steps to take to obtain CA signed certificates and how to incorporate them into the required RACF or HFS keyrings.

Scenario 1

We set out to create a self-signed server certificate, import this server certificate into the FTP client’s Trusted Authorities database for our PC-based client (WS_FTP Pro), and into the RACF keyring of our z/OS based server. This scenario shows both the **gskkyman** and RACF certificate generation. For this scenario, no Client Authentication was used.

We created a self-signed server certificate using **gskkyman** as shown in Figure 11-5 on page 261. This certificate was created in our key database file ftpv1r2.kdb, which was also created using the **gskkyman** command in z/OS UNIX. We set the certificate to be our default certificate and requested the certificate be exported to a file. We then took this exported certificate and FTPed it to our client workstation.

```

Key database menu

Current key database is /ftpv1r2

  1 - List/Manage keys and certificates
  2 - List/Manage request keys
  3 - Create new key pair and certificate request
  4 - Receive a certificate issued for your request
  5 - Create a self-signed certificate
  6 - Store a CA certificate
  7 - Show the default key
  8 - Import keys
  9 - Export keys
 10 - List all trusted CAs
 11 - Store encrypted database password

  0 - Exit program

Enter option number (or press ENTER to return to the parent menu): 5
Enter version number of the certificate to be created (1, 2, or 3) [3]: 3
Enter a label for this key.....> FTP Server certificate
Select desired key size from the following options (512):
  1: 512
  2: 1024
Enter the number corresponding to the key size you want: 2
Enter certificate subject name fields in the following.
  Common Name (required).....> ITSO.RALEIGH.IBM.COM
  Organization (required).....> IBM
  Organization Unit (optional).....> ITSO RALEIGH TP SERVER
  City/Locality (optional).....> Raleigh
  State/Province (optional).....> NC
  Country Name (required 2 characters)..> US
Enter number of valid days for the certificate [365]:
Do you want to set the key as the default in your key database? (1 = yes, 0 = no
) Y1": 1
Do you want to save the certificate to a file? (1 = yes, 0 = no) [1]: 1
Should the certificate binary data or Base64 encoded ASCII data be saved? (1 = A
SCII, 2 = binary) [1]: 1
Enter certificate file name or press ENTER for "cert.arm": ftpv1r2


Please wait while self-signed certificate is created...

Your request has completed successfully, exit gskkyman? (1 = yes, 0 = no) [0]: 1

```

Figure 11-5 gskkyman generating a self-signed FTP server certificate

WS_FTP Pro client

1. Using our WS_FTP Pro FTP client, we imported this file into its Trusted Authorities database . Figure 11-6 on page 262 shows the result of our certificate import.

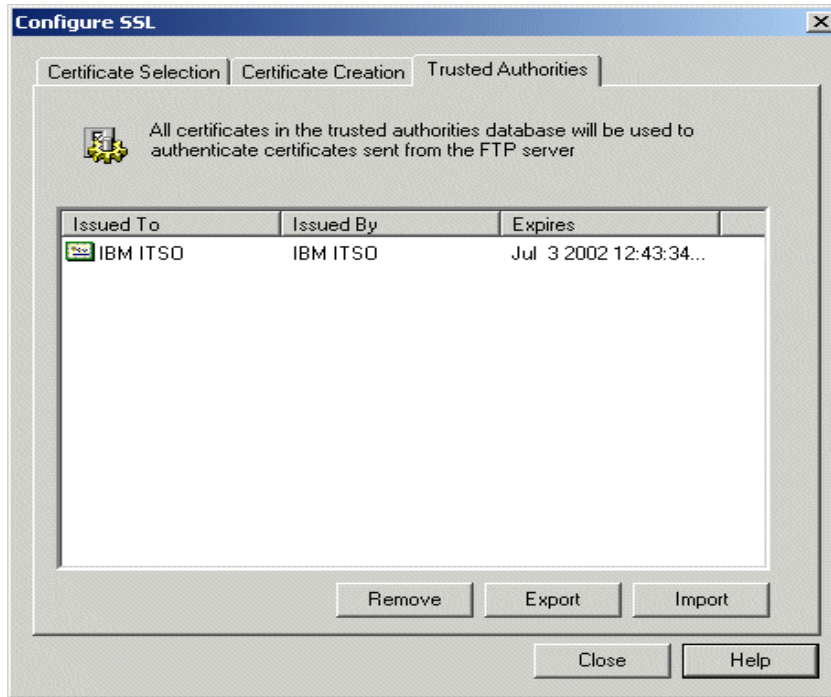


Figure 11-6 Panel showing our digital certificate

- We then set up our WS_FTP client with the Secure (SSL) option selected and the relevant FTP server ip_address, user ID, and password as shown in Figure 11-7.

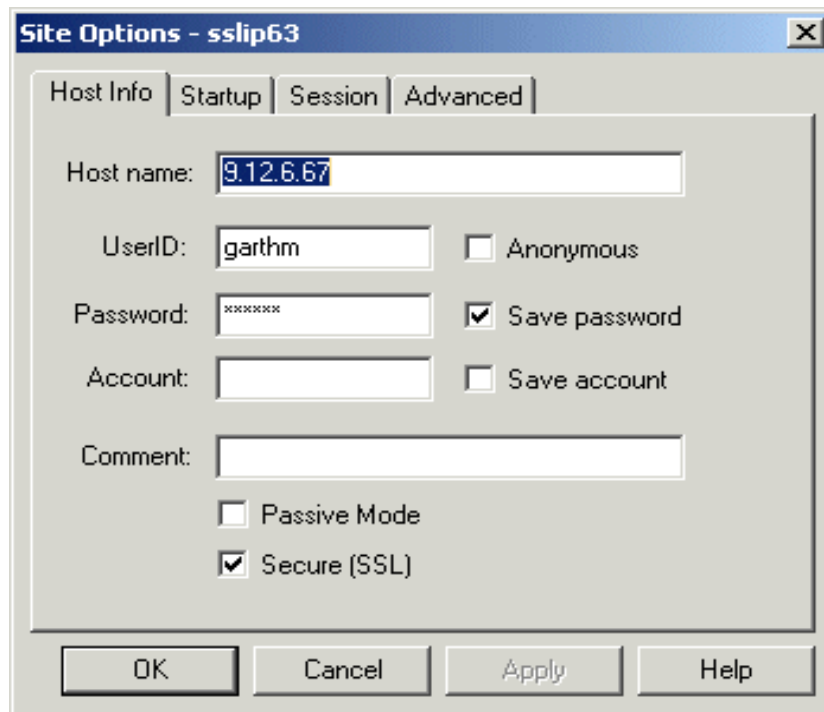


Figure 11-7 Panel showing host and SSL selection

3. On the FTP server side, we set up FTP.DATA parameters for the server as shown in Figure 11-8. Our parameter choices were as follows:

1 EXTENSIONS AUTH_TLS

Specifies that the TLS authentication is supported. This means that the server supports receiving the AUTH command with values TLS, TLS-C, TLS-P, and SSL.

2 SECURE_FTP REQUIRED

Specifies that authentication is required. This means that the FTP server must receive an AUTH command before the client can log in.

3 SECURE_LOGIN NO_CLIENT_AUTH

Specifies that the FTP server does not request the client certificate.

4 SECURE_DATACONN PRIVATE

The data connection is required to be integrity protected and is required to be encrypted. The client must issue a valid AUTH command before attempting to log in to the FTP server. The server accepts the PROT P command.

5 KEYRING *keyring*

In this statement, *keyring* is the name of the keyring. If the parameter starts with a slash (/), it is the name of an HFS file. Otherwise, it is the name of a RACF resource that defines the keyring. In our case, we pointed to our HFS key database we created earlier.

```

DEBUG SEC
DEBUG SOC(3)
EXTENSIONS AUTH_TLS           ;Support TLS authentication
SECURE_FTP REQUIRED            ;Server AUTH LVL 1
SECURE_LOGIN NO_CLIENT_AUTH   ;do not request client cert.
SECURE_DATACONN PRIVATE      ;do not secure data connection
KEYRING /ftpv1r2.kdb         ;cert.

```

Figure 11-8 Server FTP.DATA security parms

4. We were now in a position to connect to our z/OS FTP server for a secure SSL session using only a self-signed server certificate and not requesting client authentication.

Z/OS client

1. Using the z/OS FTP client we now had to point our client's FTP.DATA file to a key database or keyring containing our server certificate. In this case it was HFS file /ftpv1r2.kdb (the key database created as part of **gskkyman**).

FTP.DATA parameters for our z/OS client as shown in Figure 11-9 on page 264:

1 SECURE_MECHANISM TLS

Specifies that TLS is the security mechanism that is used by the client when it sends an AUTH command.

2 SECURE_FTP REQUIRED

Specifies that authentication is required. This means that the FTP client must send an AUTH command before the client can log in.

3 SECURE_DATACONN PRIVATE

The data connection is required to be integrity protected and is required to be encrypted. The client must issue a valid AUTH command before attempting to log in to the FTP server. The client allows the PROTECT PRIVATE or PRIVATE subcommands.

– 4 KEYRING *keyring*

In this statement, *keyring* is the name of the keyring. If the parameter starts with a slash (/), it is the name of an HFS file. Otherwise, it is the name of a RACF resource that defines the keyring.

```
RDW          false      ; Do not retain RDWs as data
SECURE_MECHANISM TLS
SECURE_FTP REQUIRED
SECURE_DATACONN PRIVATE
keyring /ftpv1r2.kdb
```

Figure 11-9 Client FTP.DATA security parms

2. We started our FTP session and were able to connect successfully as can be seen in Figure 11-10.

```
220-FTPD1 IBM FTP CS V1R2 at wtsc63oe.itso.ibm.com, 13:25:59 on 2002-05-15.
220 Connection will close if idle for more than 5 minutes.
>>> AUTH TLS
234 Security environment established - ready for negotiation
Authentication negotiation succeeded
>>> PBSZ 0
200 Protection buffer size accepted
>>> PROT P
200 Data connection protection set to private
Data connection protection is private
NAME (9.12.6.67:GARTHM):
garthm
>>> USER garthm
331 Send password please.
PASSWORD:

>>> PASS
230-Processing FTPS.RC configuration file - GARTHM.FTPS.RC
230-SITE command was accepted
230-No users are allowed to use SITE DEBUG
230-SITE command was accepted
230-"GARTHM.FTP.CNF1." is the working directory name prefix.
***
```

Figure 11-10 TLS session setup using z/OS client

Even though we did not include any ciphers to be used as part of the CIPHERSUITE FTP.DATA parameter, it is possible for the client or server to select which ones to use. Refer to *z/OS V1R2.0 CS: IP Configuration Reference*, SC31-8776 for a complete list of all the available Ciphersuite parameters. By including the Ciphersuite parameter in FTP.DATA, the client and server specify the list of encryption types that they support. The client and server negotiate which of the available ciphers is used for the data encryption by specifying the desired ciphers in order of preference. The actual cipher used is the best match between what the server supports and what the client requests. If the server does not support any of the ciphers the client requests, the TLS handshake will fail and the connection will be closed.

We repeated the above exercise using RACF instead of **gskkyman**. We only used our z/OS FTP client to demonstrate this scenario, as illustrated in Figure 11-11 on page 265.

We started by creating an FTP server certificate for use by our FTPDB1 server. We used the RACDCERT command embedded in JCL executing the batch TSO IKJEFT01 program. The RACDCERT commands can be executed from batch, the TSO command line, and by using the RACF ISPF panels. All RACDCERT commands will be shown on their own from now on and not as part of a JCL job.

Referring to Figure 11-11:

- ▶ **1** Defining the server certificate using RACDCERT requires the RACF ID used by the FTP server.
- ▶ **2** The WITHLABEL parameter is important when connecting this certificate to other keyrings as shown in the LABEL (**5**) parameter of the CONNECT statement that follows.
- ▶ After we defined the certificate we created our server's keyring. In this example, we chose the name FTPRing **3**.
- ▶ Next we connected our server certificate with LABEL('FTPb Server') **5** to our keyring FTPRing **3**.
- ▶ We then exported the FTP server certificate to an MVS data set **6**, which can be used to export to an FTP client for entry into the client's key database as a Trusted CA certificate. It was not required for us to include this final RACDCERT command to create an MVS data set, since our z/OS client and server formed part of a shared RACF database.

```

//GARTH1 JOB 'SET UP TN3270 CERT','GARTHM',CLASS=A,MSGCLASS=X
//CERTAUTH EXEC PGM=IKJEFT01,DYNAMNBR=30,REGION=4096K
/**
/** Add the top-level self-signed certificate for the certificate
/** authority (ourselves)
/**
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *
RACDCERT ID(TCPIPMS) GENCERT SUBJECTSDN(CN('ITSO.RALEIGH.IBM.COM') - 1
      O('IBM Corporation') -
      OU('ITSO Raleigh FTPB Server') -
      C('US')) -
WITHLABEL('FTPb Server') 2
RACDCERT ID(TCPIPMS) ADDRING(FTPRing) 3
RACDCERT id(TCPIPMS) CONNECT(id(TCPIPMS) - 4
      LABEL('FTPb Server') - 5
      RING(FTPRing) - 3
      DEFAULT - 7
      USAGE(PERSONAL))
RACDCERT id(TCPIPMS) EXPORT(label('FTPb Server')) - 6
      FORMAT(CERTB64) DSN('FOCAS.RACDCERT.FTPSERV.CERT')
/**
//

```

Figure 11-11 Creating a server certificate, adding it to a ring and exporting it to a data set

- ▶ **7** The DEFAULT statement is imperative because SSL has to know which certificate to pass to the client. Figure 11-12 shows the error in an FTP trace if the server certificate is not defined in the RACF keyring as the DEFAULT certificate.

```

authClient: init failed with rc = 403 (GSK_ERR_NO_CERTIFICATE)

```

Figure 11-12 Error received when server certificate was not default in FTPRing

Use the z/OS console `d a` command as shown in Figure 11-13 to obtain the correct RACF user ID for the FTP started task.

```
d a,ftpdb1
RESPONSE=SC63
IEE115I 11.46.40 2002.144 ACTIVITY 433
  JOBS      M/S      TS USERS    SYSAS    INITS    ACTIVE/MAX VTAM    OAS
00008      00043     00010     00030    00022    00010/00030    00023
  FTPDB1    STEP1      TCPIMVS  OWT  AO  A=0055  PER=NO  SMC=000
                                     PGN=N/A  DMN=N/A  AFF=NONE
                                     CT=000.041S  ET=018.966S
                                     WUID=STC17984  USERID=TCPIMVS
```

Figure 11-13 Displaying the FTP server ID

At this stage, we had an FTP server certificate and a keyring called FTPRing containing this server certificate. We now wanted our z/OS client (GARTHM) to include as part of its keyring our server certificate. Figure 11-14 shows how we created a keyring for the client. In the example, we have shown that you can create a keyring with an identical name to an existing one (in this case it is the same as the FTP server's keyring) as long as it is associated with a different user ID. Once the keyring is created, we connect the server certificate's CA (in this case the server certificate is self-signed, so the server's CA certificate is the server's certificate) to the client's keyring.

Note: Even though in this example the client and server both use FTPRing as the keyring name, they do not have to be the same. Remember that the keyring name is specified in the ftp.data file. This file can be different for server and client.

```
RACDCERT ID(garthm) ADDRING(FTPRing)
RACDCERT ID(garthm) CONNECT(ID(TCPIMVS) -
                                LABEL('FTPb Server') -
                                RING(FTPRing) -
                                USAGE(PERSONAL))
```

Figure 11-14 Including our server certificate into our client keyring

All that was left for us to do was ensure our client's FTP.DATA file contained the relevant security parameters. The FTP.DATA file we used for our server is the same as the one displayed in Figure 11-8 on page 263. Our client FTP.DATA file is shown in Figure 11-15.

```
Filetype      SEQ      ; File Type = SEQ (default)
RDW           false    ; Do not retain RDWs as data
SECURE_MECHANISM TLS
SECURE_FTP REQUIRED
SECURE_DATACONN PRIVATE
keyring FTPRing
```

Figure 11-15 Client FTP.DATA file

Scenario 2

First we create a server certificate and import this server certificate into the FTP client's Trusted Authorities database for a PC-based client (Cute) and into RACF-based client keyring of a z/OS V1R2 FTP client. Then we create a client certificate and import this certificate into the z/OS FTP server's keyring with CERTAUTH authority.

We set up a client certificate to demonstrate the function of a server wanting an identifying client certificate. We did this by navigating to the Configure SSL window shown in Figure 11-16 via the Tools selection tab on the main menu.

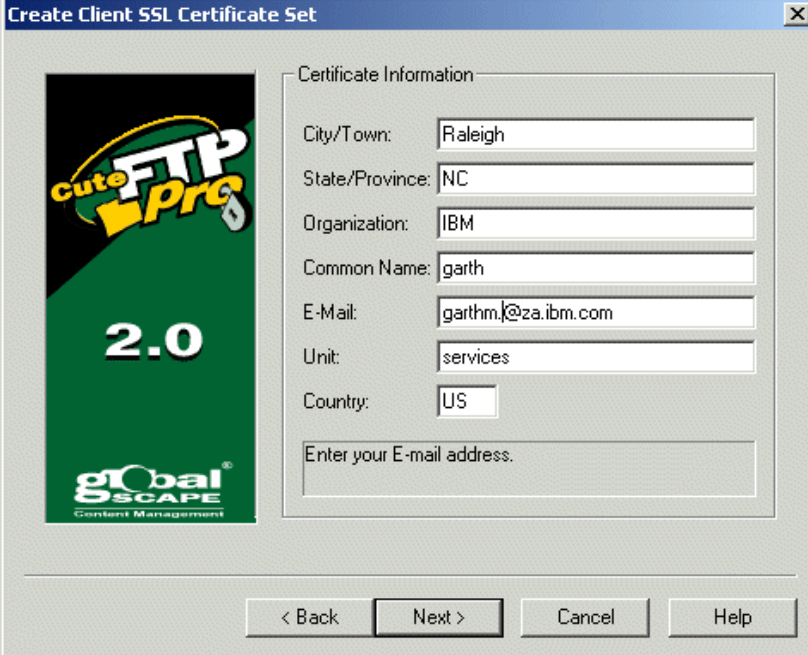


Figure 11-16 Cute_FTP Pro client certificate generation

We then entered the following information:

City/Town	Raleigh (the city where we were located)
State/Province	NC (North Carolina the state or province in which Raleigh lies)
Organization	IBM (the company or individual user name)
Common Name	garth (this can be either the name of the person creating the certificate or the fully qualified domain name of the server associated with the host)
E-mail	garthm@za.ibm.com (e-mail address of the certificate owner)
Unit	services (department)
Country	US (the country for Raleigh, NC)

Once we completed the form and created the certificate, it generated the keys, certificate, and certificate signing request. Figure 11-17 on page 268 shows the three files that were generated as a result of this process.

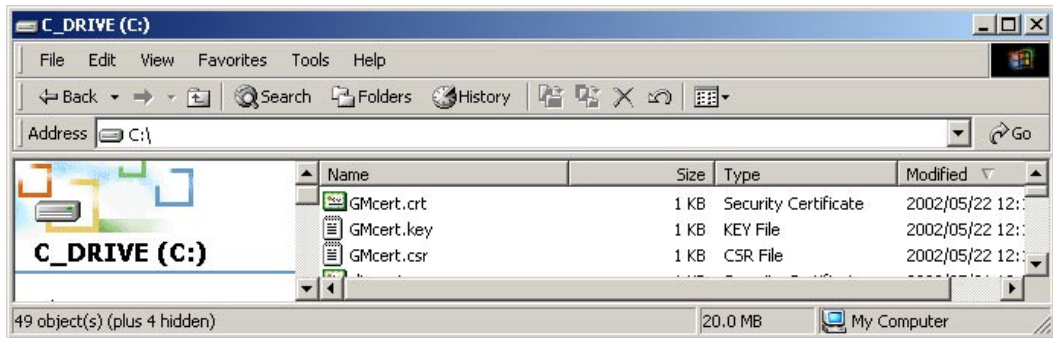


Figure 11-17 Files generated from client certificate generation process

We then FTPed the created certificate 'GMcert.crt' to z/OS for insertion into the FTP servers keyring.

There are some important things to note while FTPing this certificate to z/OS. First, you have to save the certificate with RECFM of variable blocked (VB). Issue **quote site recfm=vb** from the client to achieve this. Also we found that some FTP clients do not conform to RFC959, which in a nutshell states that when the data type is set to ASCII, the receiver knows that the data is character data and that each line of data is terminated via a control sequence of Carriage Control plus Line Feed (CRLF). This is, in ASCII, X'0D0A' for DOS and OS/2 systems and just LF (line feed) X'0A' for UNIX systems. They seem to adopt the UNIX system standard, which is frowned upon by the z/OS or in fact any OS/390 system. The certificate ends up as one long record and is normally truncated if the file's record length is smaller than the record being inserted.

Another thing to consider while displaying the certificate is that it would display properly when viewed with Wordpad or even DOS EDIT, but Notepad shows them all funny. The EDIT command in DOS does, however, change the 'LF' to CRLF' when you save the edited file.

Figure 11-18 on page 269 show the partial contents of our self-signed digital certificate GMCERT.CRT. It also shows the hexadecimal representation of this file and what it should be like when using FTP to transfer the file to z/OS.

```

:First 4 lines of Certificate
-----BEGIN CERTIFICATE-----
MIICiDCCAfGgAwIBAAIBADANBgkqhkiG9w0BAQQFADCBiTElMAkGA1UEBhMCVVMx
CzAJBgNVBAGTAK5DMQwwCgYDVQQKEwNJQk0xEDA0BgNVBACtB1JhbGVpZ2gxETAP
BgNVBAsTCHN1cnZpY2VzMR8wHQYJKoZIhvcNAQkBFhBqYW11c0B1cy5pYm0uY29t
:
:File in HEX
2D2D2D2D2D424547494E2043455254494649434154452D2D2D2D2D0A4D494943694443434166476741774942
414149424144414E42676B71686B69473977304241515146414443426954454C4D416B474131554542684D43
56564D780A437A414A42674E5642416754416B35444D517777436759445651514B45774E4A516B3078454441
4F42674E564241635442314A68624756705A326778455441500A42674E564241735443484E6C636E5A705932
567A4D5238774851594A4B6F5A496876634E41516B42466842715957316C6330423163793570596D30755932
39740A
:
:DOS EDITED and SAVED file
2D2D2D2D2D424547494E2043455254494649434154452D2D2D2D2D0D0A4D4949436944434341664767417749
42414149424144414E42676B71686B69473977304241515146414443426954454C4D416B474131554542684D
4356564D780D0A437A414A42674E5642416754416B35444D517777436759445651514B45774E4A516B307845
44414F42674E564241635442314A68624756705A326778455441500D0A42674E564241735443484E6C636E5A
705932567A4D5238774851594A4B6F5A496876634E41516B42466842715957316C6330423163793570596D30
75593239740D0A

```

Figure 11-18 Client certificate to be FTPed to z/OS

After FTPing our certificate to z/OS, a RECFM VB data set named GARTH.MGCERT (2), we added this certificate to the FTP server's ID (TCPIPMS) with a label named "FTP client garth" as can be seen in Figure 11-19. It was also added as a trusted certificate with the inclusion of the TRUST (3) parameter. To find the name of the label for the server certificate we issued the RACDCERT command displayed in Figure 11-20 on page 270. So in effect we are connecting our client certificate to our FTP server's RACF keyring called FTPring. This keyring called FTPring is the one referred to in FTP.DATA. This keyring was created when we created our server certificate and is shown in Figure 11-11 on page 265.

```

RACDCERT ID(FOCAS) ADD('GARTH.MGCERT') 2-
          TRUST 3 WITHLABEL('FTP client garth')
RACDCERT ID(TCPIPMS) CONNECT(ID(FOCAS) -
          LABEL('FTP client garth') -
          RING(FTPring) -
          USAGE(PERSONAL))

```

Figure 11-19 RACDCERT commands to add the client certificate

To check the result of all our RACDCERT commands we then displayed which certificates were present in our keyrings. We started by displaying our FTP server keyring as listed in Figure 11-20 on page 270. This display shows the presence of our server certificate (1) and our client certificate (2).

Note: 1 Our server certificate is set as the DEFAULT in our server keyring FTPring.

```

racdcert id(tcpipmvs) listr(FTP Ring)
Digital ring information for user TCPIPMVS:

Ring:
  >FTP Ring<
Certificate Label Name      Cert Owner      USAGE      DEFAULT
-----
FTPb Server                ID(TCPIPMVS)   PERSONAL   YES      1
FTP client garth          ID(FOCAS)      PERSONAL   NO       2

```

Figure 11-20 Listing the FTP server keyring

At this stage we had the following:

- ▶ A client certificate on our Cute FTP Pro client
- ▶ A server certificate on our FTP server user ID (TCP/IPMVS) keyring
- ▶ A client certificate on our FTP server user ID(TCP/IPMVS) keyring

We still, however, required a server certificate in our client's Trusted Certificates database. To do this, we exported our server certificate to an MVS data set, FTPed it to our client, and imported it to our Cute FTP Pro Client's Trusted Certificates database. If you are using the same client as in scenario 1, the server certificate should already be there. If this server certificate is not already present in the Client Trusted Certificates database you can export the certificate as shown in Figure 11-21. This file has to be FTPed to the client PC.

```

RACDCERT id(TCP/IPMVS) EXPORT(label('FTPb Server')) -
          FORMAT(CERTB64) DSN('GARTH.M.RACDCERT.FTPSERV.CERT')

```

Figure 11-21 Exporting a server certificate from RACF

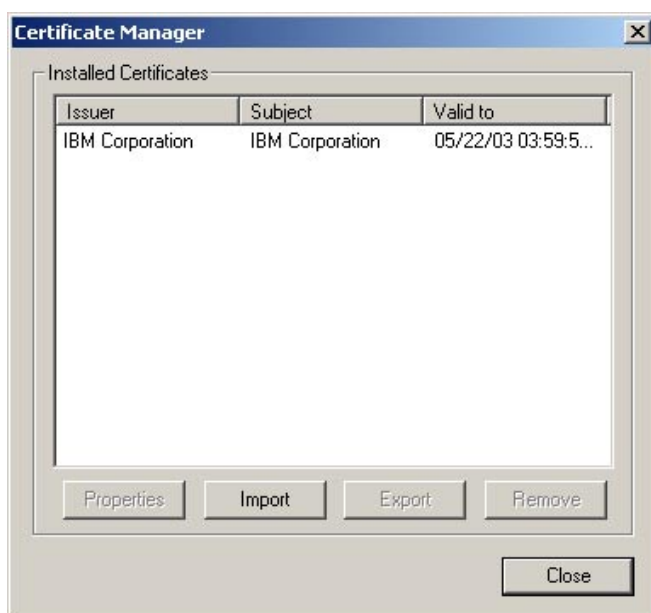


Figure 11-22 Cute FTP Pro Trusted Certificates database

All that was left for us to do was to point to our client to our certificate “gmcert.crt”, as can be seen in Figure 11-23. This product also required us to point to a key that it generated as part of the certificate generation process.

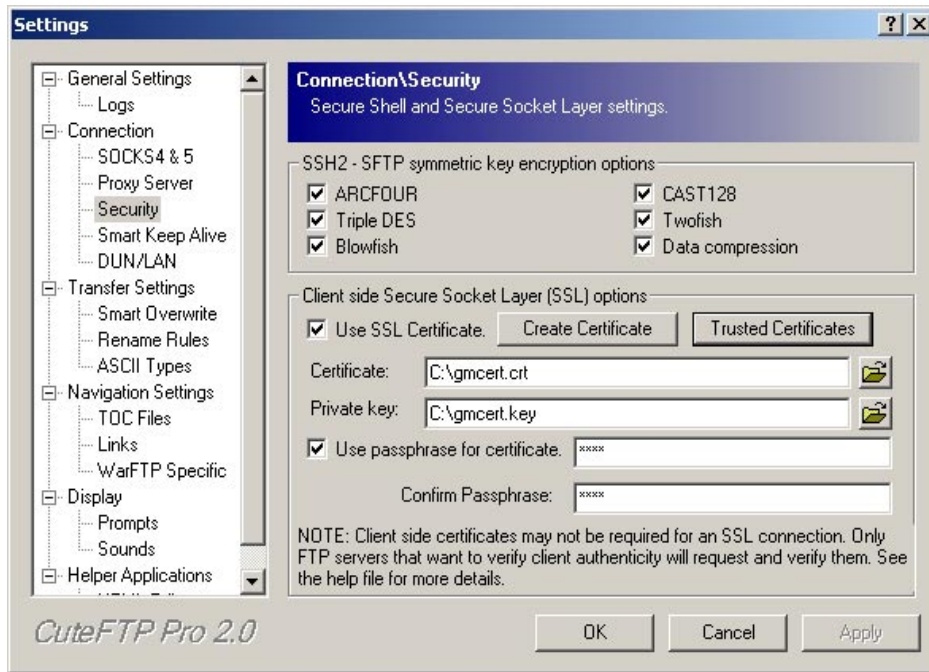


Figure 11-23 Client certificate selection window

The final task was then to set up an SSL session and connect to our z/OS FTP server. We restarted our z/OS FTP server with our FTP.DATA having the relevant security parameters as can be seen in Figure 11-24. The pertinent parameter being SECURE_LOGIN VERIFY_USER, which ensures that the server requests the client’s certificate.

```

DEBUG SEC
DEBUG SOC(3)
EXTENSIONS AUTH_TLS           ;Support TLS authentication
SECURE_FTP ALLOWED            ;Server allows AUTH cmdnd
SECURE_LOGIN VERIFY_USER      ;request client cert.
SECURE_DATACONN PRIVATE       ;secure data connection
KEYRING FTPring               ;RACF keyring FTPring.

```

Figure 11-24 FTP for scenario 2

We successfully established our session, as can be seen in Figure 11-25 on page 272.

2. Our second task was to create a client certificate (1), create a client keyring (2) and add the client(3) and server(4) certificates to this keyring. The RACDCERT statements to achieve this are shown in Figure 11-26. Use the RADCERT LIST command detailed in Figure 11-29 on page 274 to ascertain the label for the server FTPRing for ID TCPIPMVS.

```

RACDCERT id(garthm) gencert          - 1
  subjectsdn( o('IBM Corporation')  -
             cn('garthm self-signed ftp cli') -
             ou('ITSO Certificate Authority') -
             C('US'))                -
  WITHLABEL('garthm ss cli cert') trust

RACDCERT ID(garthm)  ADDRING(FTPRing)          2

RACDCERT ID(garthm)  CONNECT(ID(garthm)       - 3
                          LABEL('garthm ss cli cert') -
                          RING(FTPRing)         -
                          default              -
                          USAGE(personal))

RACDCERT ID(garthm)  CONNECT(ID(tcpipmvs)     - 4
                          LABEL('FTPb Server') -
                          RING(FTPRing)         -
                          USAGE(certauth))

```

Figure 11-26 Creating client certificate and keyring - FTPRing

3. At this stage we had only a server certificate within our server's keyring and had to add our client certificate to this keyring as well. This can be seen in Figure 11-27. Here we see our client certificate label "garthm ss cli cert" 1 being connected to our FTP server's user ID "TCPIPMVS" 2 keyring "FTPRing" 3.

```

RACDCERT ID(tcpipmvs) 2 CONNECT(ID(garthm)    -
                              LABEL('garthm ss cli cert') - 1
                              RING(FTPRing)     - 3
                              USAGE(certauth))

```

Figure 11-27 Add client certificate to server FTPRing

To ensure everything was set up as planned, we used the RACDCERT command to display things. As can be seen in Figure 11-28, both our client and server certificates are present in our server's keyring.

```

racdcert id(tcpipmvs) listring(FTPRing)

Digital ring information for user TCPIPMVS:

Ring:
>FTPRing<
Certificate Label Name      Cert Owner      USAGE      DEFAULT
-----
FTPb Server                 ID(TCPIPMVS)   PERSONAL   YES
FTP client garth            ID(FOCAS)      PERSONAL   NO
garthm ss cli cert         ID(GARTHM)     CERTAUTH   NO

***

```

Figure 11-28 RACDCERT ID(TCPIPMVS) listring(FTPRing) command output

Figure 11-29 shows the server's digital certificate. This display also shows the label "FTPb Server", which is important to know when attaching client certificates, because this label has to be used in the RACDCERT command.

```

racdcert id(tcipmvs) list
Digital certificate information for user TCIPMVS:

Label: FTPb Server
Certificate ID: 2Qjjw9fJ19T14sbn14JA4oWZpYWZ
Status: TRUST
Start Date: 2002/05/21 00:00:00
End Date: 2003/05/21 23:59:59
Serial Number:
    >00<
Issuer's Name:
    >CN=ITSO.RALEIGH.IBM.COM.OU=ITSO Raleigh FTPB Server.O=IBM Corporation<
    >.C=US<
Subject's Name:
    >CN=ITSO.RALEIGH.IBM.COM.OU=ITSO Raleigh FTPB Server.O=IBM Corporation<
    >.C=US<
Private Key Type: Non-ICSF
Private Key Size: 1024
Ring Associations:
  Ring Owner: TCIPMVS
  Ring:
    >FTPRing<
  Ring Owner: GARTHM
  Ring:
    >FTPRing<

```

Figure 11-29 RACDCERT ID(TCIPMVS) list command output

To display the contents of our client user ID, GARTHM, we issued the RACDCERT LISTRING command this time with an ID of GARTHM. The output for this command is shown in Figure 11-30. The output clearly shows that we do have both our client and server certificates present in GARTHM's keyring called FTPRing.

```

racdcert id(garthm) listring(FTPRing)
Digital ring information for user GARTHM:

Ring:
  >FTPRing<
Certificate Label Name          Cert Owner    USAGE        DEFAULT
-----
garthm ss cli cert             ID(GARTHM)   PERSONAL     YES
FTPb Server                     ID(TCIPMVS) CERTAUTH     NO

***

```

Figure 11-30 RACDCERT ID(GARTHM) listring(FTPRing) command output

As a final display, Figure 11-31 on page 275 shows our client(GARTHM) certificate. Here we can see that the keyring(FTPRing) for GARTHM and the keyring(FTPRing) for TCIPMVS are connected to this certificate.

```

raccert id(garthm) list
Digital certificate information for user GARTHM:

Label: garthm ss cli cert
Certificate ID: 2QbHwdnjyNSHgZmjiJRAoqJAg50JQIOFmaNA
Status: TRUST
Start Date: 2002/05/21 00:00:00
End Date: 2003/05/21 23:59:59
Serial Number:
    >00<
Issuer's Name:
    >CN=garthm self-signed ftp cli.OU=ITSO Certificate Authority.0=IBM Cor<
    >poration.C=US<
Subject's Name:
    >CN=garthm self-signed ftp cli.OU=ITSO Certificate Authority.0=IBM Cor<
    >poration.C=US<
Private Key Type: Non-ICSF
Private Key Size: 1024
Ring Associations:
    Ring Owner: GARTHM
    Ring:
        >FTPRing<
    Ring Owner: TCPIPMVS
    Ring:
        >FTPRing<

```

Figure 11-31 RACDCERT ID(GARTHM) list command output

The FTP.DATA file we used is shown in Figure 11-24 on page 271. Our client (GARTHM) allocated the following FTP.DATA file for use with his FTP session.

```

Filetype      SEQ      ; File Type = SEQ (default)
RDW           false   ; Do not retain RDWs as data
SECURE_MECHANISM TLS
SECURE_FTP REQUIRED
SECURE_DATACONN PRIVATE
keyring FTPRing

```

Figure 11-32 Client GARTHM's FTP.DATA file

Note: We used the same keyring name, FTPRing, for our client (GARTHM) and our z/OS FTP server (FTPDB1 - id(TCPIPMVS)). The keyring names do *not* have to be the same because they are referenced from separate FTP.DATA files.

Scenario 3

This scenario uses the z/OS client to demonstrate the adding of the SERVAUTH class to provide Authentication Level 3 security

By activating the SERVAUTH class we were able to obtain a more granular level of access control. If the installation has activated the SERVAUTH class and provided a profile for the port in the SERVAUTH class, only users specified in the profile are allowed to connect into the port. The resource name would be of the form:

EZB.FTP.<systemname>.<ftpdaemonname>.PORTxxxx

We then added our Profile EZB.FTP.SC63.FTPDB1.PORT21 using the RACF panels, navigating via GENERAL RESOURCE PROFILES and Add a Profile panel selections. Once we added our profile using CLASS - SERVAUTH, we tried logging on to the SC63 FTP server with the knowledge that no users were added to this profile. The FTP server was started with the FTP.DATA correct SECURE_LOGIN parameter of VERIFY_USER **1**, which can be seen in Figure 11-33.

```

DEBUG SEC
DEBUG SOC(3)
EXTENSIONS AUTH_TLS           ;Support TLS authentication
SECURE_LOGIN ALLOWED         ;Server allows AUTH cmdnd
SECURE_LOGIN VERIFY_USER    ;DO NOT REQUEST CLIENT CERT.  1
SECURE_DATACONN PRIVATE     ;secure data connection
KEYRING FTPRing             ;RACF keyring FTPRing.

```

Figure 11-33 Server FTP.DATA options showing SECURE_LOGIN VERIFY_USER

At this point no users were added to the SERVAUTH class EZB.FTP.SC63.FTPDB1.PORT21, which therefore resulted in this session being rejected. On the client side, the logon sequence proceeded until the password was entered. Figure 11-34 shows the 530 PASS command failed error. A RACF authority violation error was shown on the system console, seen in Figure 11-35.

```

FTP: using TCPIP
Connecting to: 9.12.6.67 port: 21.
220-FTPDB1 IBM FTP CS V1R2 at dns63.zos12.ra1.ibm.com, 15:53:50 on 2002-05-22.
220 Connection will close if idle for more than 5 minutes.
>>> AUTH TLS
234 Security environment established - ready for negotiation
Authentication negotiation succeeded
>>> PBSZ 0
200 Protection buffer size accepted
>>> PROT P
200 Data connection protection set to private
Data connection protection is private
NAME (9.12.6.67:GARTHM):
garthm
>>> USER garthm
331 Send password please.
PASSWORD:

>>> PASS
530 PASS command failed
Command:

```

Figure 11-34 Client session rejection message

```

do AUTHTRC for EZB.FTP.SC63.FTPDB1.PORT21
ICH408I USER(GARTHM ) GROUP(SYS1 ) NAME(GARTH MADELLA ) 261
EZB.FTP.SC63.FTPDB1.PORT21 CL(SERVAUTH)
INSUFFICIENT ACCESS AUTHORITY
ACCESS INTENT(READ ) ACCESS ALLOWED(NONE )

```

Figure 11-35 User logon attempt without access to SERVAUTH profile

We then added user GARTHM to profile EZB.FTP.SC63.FTPDB1.PORT21 and user GARTHM was able to logon successfully to this secure session.

11.1.3 FTP using Kerberos

For a discussion of the Kerberos protocol, see 9.3.1, “Kerberos protocol overview” on page 192. For Kerberos z/OS implementation, see 9.3.4, “Kerberos implementation in z/OS” on page 198. Kerberos is implemented using the GSSAPI and Kerberos APIs provided by the z/OS Security Server. With the FTP server, however, Kerberos is implemented using the GSSAPI mechanism.

Kerberos principal definition

Every user and process that authenticates with Kerberos must have a *principal* name. For the FTP server, the principal name is “ftp” (lowercase). As outlined in 9.3.4, “Kerberos implementation in z/OS” on page 198, you add Kerberos principals to the KDC by defining a KERB segment on a user’s RACF profile. You can use any RACF user, but most often it would be the FTP started task owner’s RACF ID; this is not a requirement however.

Figure 11-36 shows a two-step process to add the FTP server’s Kerberos principal. In the RACF ADDUSER command, a new user is added. The second step is to use the RACF ALTUSER command to set the password to NOEXPIRED and add the KERB segment with the correct Kerberos principal name of “ftp”. The reason the KERB segment is not just added on the ADDUSER command is that the Kerberos password is generated only when you change the password; thus the ALTUSER command is needed.

```
ADDUSER KERBTST1 DFLTGRP(SYS1) PASSWORD(TEMPPASS)
ALTUSER KERBTST1 PASSWORD(kerberos) NOEXPIRED      -
      KERB(KERBNAME(ftp))
```

Figure 11-36 Adding a new user and creating a KERB segment on that user for the FTP server

FTP server configuration statements

To customize the FTP server for Kerberos authentication, the following values have to be set in FTP.DATA, as shown in Figure 11-37.

```
Filetype      SEQ      ; File Type = SEQ (default)
RDW           false    ; Do not retain RDWs as data
EXTENSIONS AUTH_GSSAPI 1
SECURE_FTP REQUIRED 2
SECURE_LOGIN REQUIRED 3
SECURE_CTRLCONN PRIVATE 4
SECURE_DATACONN PRIVATE 5
```

Figure 11-37 FTP.DATA server statements for Kerberos

The following can be coded:

► **EXTENSIONS AUTH_GSSAPI 1**

Specifies that Kerberos authentication type is supported using the GSSAPI. The server will support receiving the AUTH GSSAPI command but does not require it to be sent.

► **SECURE_FTP REQUIRED 2**

Specifies that authentication is required. The server requires receiving the AUTH GSSAPI command before the user can log on.

- ▶ **SECURE_FTP ALLOWED**
This is the default value and specifies that authentication is supported but not required.
- ▶ **SECURE_LOGIN VERIFY_USER**
Connections are only allowed for users who can authenticate via the FTP AUTH mechanism and whose user ID matches the user name in the Kerberos credentials supplied by the client on the ADAT command. Anonymous FTP may also be allowed if it is configured. If this option is used, ensure your Kerberos principal name is entered in uppercase on your Kerberos segment in the user's RACF profile.
- ▶ **SECURE_LOGIN REQUIRED 3**
Connections are only allowed for users who can authenticate via the FTP AUTH mechanism. Anonymous FTP may also be allowed if it is configured. The user ID supplied by the client does not need to match the user name in the Kerberos credentials supplied by the client on the ADAT command.
- ▶ **SECURE_LOGIN NO_CLIENT_AUTH**
This is the default value and connections will be allowed for users who will be authenticated through the normal method (user ID/password) and also through the FTP AUTH mechanism.
- ▶ **SECURE_CTRLCONN CLEAR (default)**
The command channel is not required to be integrity protected or encrypted. The server will accept the CCC command.
- ▶ **SECURE_CTRLCONN SAFE**
The command channel is required to be integrity protected but not required to be encrypted. The client must issue a valid AUTH command before attempting to log on to the FTP server. The server will *not* accept the CCC command.
- ▶ **SECURE_CTRLCONN PRIVATE 4**
The command channel is required to be integrity protected and encrypted. The client must issue a valid AUTH command before attempting to log on to the FTP server. The server will *not* accept the CCC command.
- ▶ **SECURE_DATACONN NEVER**
The data channel is required to *not* be integrity protected *nor* encrypted. The server will *only* accept the PROT C command.
- ▶ **SECURE_DATACONN CLEAR (default)**
The data channel is not required to be integrity protected or encrypted. The server will accept the PROT command.
- ▶ **SECURE_DATACONN SAFE**
The data channel is required to be integrity protected but not required to be encrypted. The client must issue a valid AUTH command before attempting to log on to the FTP server. The server will *not* accept the PROT C command.
- ▶ **SECURE_DATACONN PRIVATE 5**
The data channel is required to be integrity protected and is required to be encrypted. The client must issue a valid AUTH command before attempting to log on to the FTP server. The server will *only* accept the PROT P command.
- ▶ **SECURE_PBSZ**

FTP client configuration for Kerberos

For the z/OS FTP client environment, Kerberos authentication is defined in the FTP.DATA file for the client as shown in Figure 11-38.

SECURE_MECHANISM GSSAPI	1
SECURE_FTP REQUIRED	2
SECURE_DATACONN PRIVATE	3
SECURE_CTRLCONN PRIVATE	4

Figure 11-38 Client FTP.DATA showing Kerberos security statements

The following can be coded:

- ▶ **SECURE_MECHANISM GSSAPI 1**
Specifies that the Kerberos authentication type is supported using the GSSAPI. This means that we will issue the AUTH GSSAPI command but do not require it to be accepted by the server.
- ▶ **SECURE_FTP REQUIRED 2**
Specifies that authentication is required. The client will issue the AUTH GSSAPI command and require it to be accepted by the server.
- ▶ **SECURE_FTP ALLOWED**
This is the default value and specifies that authentication is supported but not required.
- ▶ **SECURE_CTRLCONN CLEAR**
This option is the default where the command channel is not required to be integrity protected or encrypted. The client will allow the CPROTECT and CCC subcommands.
- ▶ **SECURE_CTRLCONN SAFE**
The command channel is required to be integrity protected but not required to be encrypted. The client must issue a valid AUTH command before attempting to log on to the FTP server. The client will *not* allow the CPROTECT CLEAR and CCC subcommands.
- ▶ **SECURE_CTRLCONN PRIVATE 4**
The command channel is required to be integrity protected and encrypted. The client must issue a valid AUTH command before attempting to log on to the FTP server. The client will *not* allow the CCC or CPROTECT [CLEAR | SAFE] subcommands.
- ▶ **SECURE_DATACONN NEVER**
The data channel is required to *not* be integrity protected *nor* encrypted. The client will *only* allow the PROTECT CLEAR or CLEAR subcommands.
- ▶ **SECURE_DATACONN CLEAR**
This is the default value where the data channel is not required to be integrity protected or encrypted. The client will allow the PROTECT, CLEAR, SAFE, and PRIVATE subcommands.
- ▶ **SECURE_DATACONN SAFE**
The data channel is required to be integrity protected but not required to be encrypted. The client must issue a valid AUTH command before attempting to log on to the FTP server. The client will *not* allow the PROTECT CLEAR or CLEAR subcommands.

▶ SECURE_DATACONN PRIVATE 3

The data channel is required to be integrity protected and is required to be encrypted. The client must issue a valid AUTH command before attempting to log on to the FTP server. The client will not allow the CLEAR, SAFE, or PROTECT [CLEAR | SAFE] subcommands.

FTP client command-line options for Kerberos

The z/OS FTP client will use Kerberos if configured as described above. You can also override the default behavior of the client with command-line options as described here.

Restriction: The FTP client must specify a host name, not an IP address, for Kerberos to work. This is because a `gethostbyname()` call is used to get the IP address of the client, which fails if an IP address is specified on the `ftp` command.

The following are options to the FTP client:

▶ FTP *dnsname* -a GSSAPI

This is the same as specifying SECURE_MECHANISM GSSAPI and SECURE_FTP ALLOWED in the FTP.DATA file. This parameter overrides the values of SECURE_MECHANISM and SECURE_FTP in the FTP.DATA file.

▶ FTP *dnsname* -a NEVER

The FTP client will *not* attempt to negotiate authentication. This parameter overrides the value of SECURE_MECHANISM in the FTP.DATA file.

▶ FTP *dnsname* -A GSSAPI

This is the same as specifying SECURE_MECHANISM GSSAPI and SECURE_FTP REQUIRED in the FTP.DATA file. This parameter overrides the value of SECURE_MECHANISM and SECURE_FTP in the FTP.DATA file.

▶ FTP *dnsname* -A NEVER

The FTP client will *not* attempt to negotiate authentication. This parameter overrides the value of SECURE_MECHANISM in the FTP.DATA file.

▶ FTP *dnsname* -x

The FTP client will attempt to set the data connection protection level to "PRIVATE" automatically after authentication negotiation succeeds.

FTP client ftp subcommands for Kerberos

When in a secure FTP session the following client subcommands can be issued:

▶ CCC

This subcommand turns off protection on the command connection. This command must be sent protected. Since turning off protection potentially allows an attacker to insert commands onto the command channel, some FTP servers may refuse to honor this command.

▶ CPROTECT CLEAR

This is the same as the CCC subcommand.

▶ CPROTECT SAFE

This subcommand sets the protection level on the command channel to "SAFE". This level applies integrity protection to commands and replies sent on the control connection. This is the initial setting for GSSAPI authenticated connections.

- ▶ **CPROTECT PRIVATE**
This subcommand sets the protection level on the command channel to "PRIVATE". This level applies integrity protection and encryption to commands and replies sent on the control connection.
- ▶ **CLEAR**
This subcommand turns off protection on the data connection.
- ▶ **SAFE**
This subcommand sets the protection level of the data connection to "SAFE". This level applies integrity protection to data transferred on the data connection.
- ▶ **PRIVATE**
This subcommand sets the protection level of the data connection to "PRIVATE". This level applies integrity protection and encryption to data transferred on the data connection.
- ▶ **PROTECT CLEAR**
This is the same as the CLEAR subcommand.
- ▶ **PROTECT SAFE**
This is the same as the SAFE subcommand.
- ▶ **PROTECT PRIVATE**
This is the same as the PRIVATE subcommand.

FTP Kerberos message flow

The following diagrams provide an overview on the various commands used to secure an FTP session with Kerberos. The first message actually manually entered by a client is the USER command.

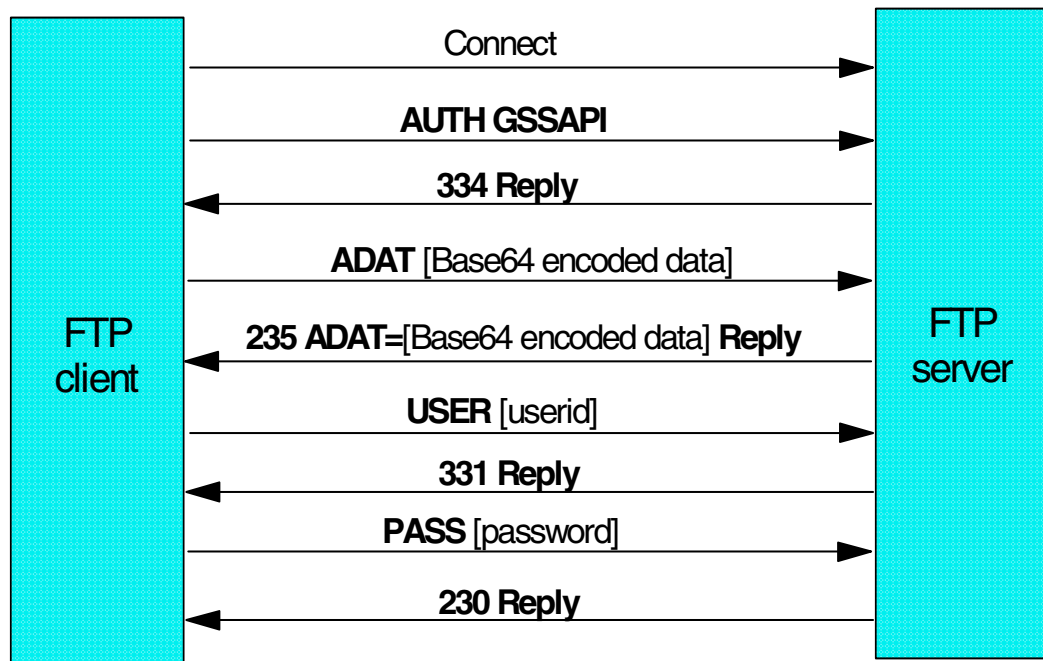


Figure 11-39 Kerberos session setup flows

As shown in Figure 11-39 on page 281, when a client establishes a protected session using Kerberos an AUTH GSSAPI flows from the client. The server responds to this AUTH command after which the client sends his credentials within the ADAT command. The ADAT gets the Kerberos credentials for the client from the Kerberos client cache. The client requests these credentials with the `kinit kerberos_client_principal` command before the FTP is started. For more information and an example of `kinit`, see 9.3.4, “Kerberos implementation in z/OS” on page 198. After the server responds positively to the ADAT command, a GSSAPI connection has been established. The client then sends a user ID and password as normal.

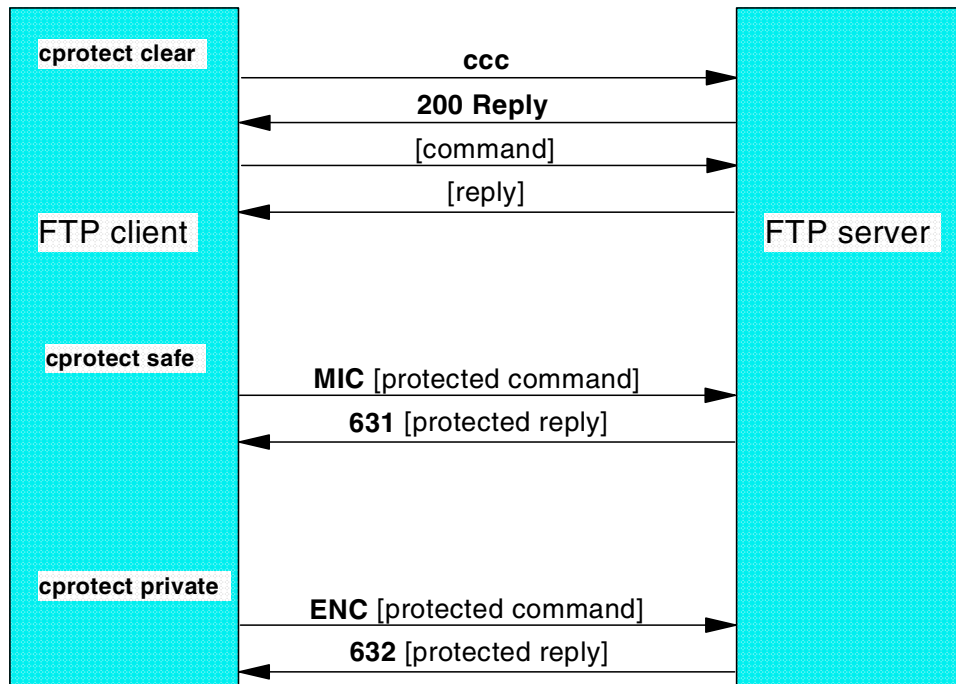


Figure 11-40 Control connection protection

The following subcommands are available:

- ▶ Client issues CPROTECT CLEAR or CCC subcommand

A CCC command is sent to the server to request authorization to send commands and replies on the control connection unprotected. If the server responds positively, then subsequent command and replies are sent unprotected.
- ▶ Client issues CPROTECT SAFE subcommand

No notification is sent to the server when the subcommand is issued. Subsequent commands are integrity protected and sent as data on a MIC command to the server. The server will integrity protect the reply and send it as data on a 631 Reply.
- ▶ Client issues CPROTECT PRIVATE subcommand

No notification is sent to the server when the subcommand is issued. Subsequent commands are integrity protected, encrypted, and sent as data on an ENC command to the server. The server will integrity protect and encrypt the reply and send it as data on a 632 Reply.

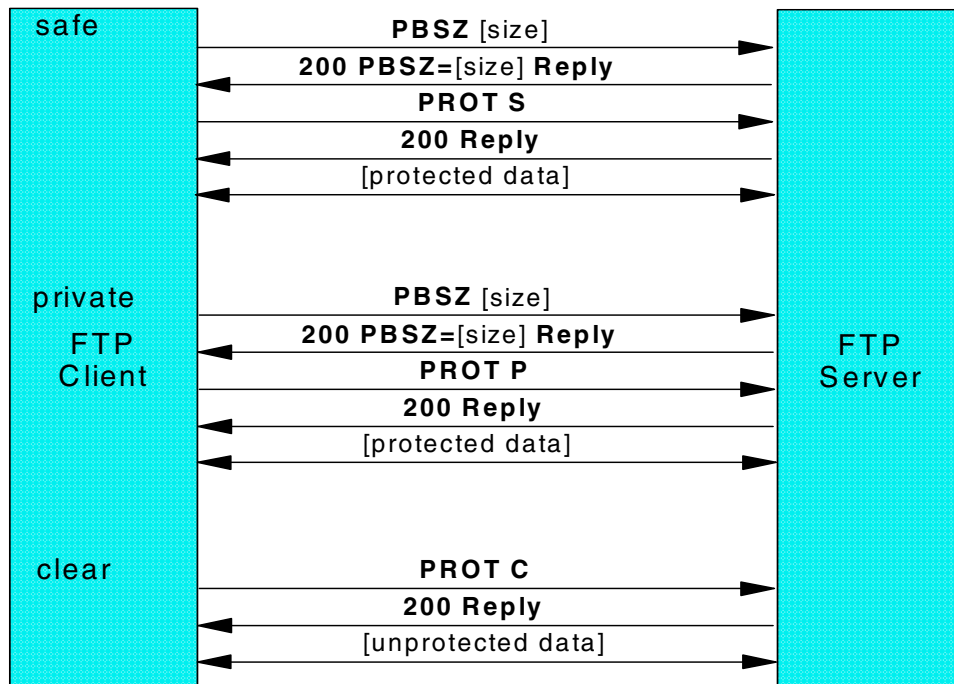


Figure 11-41 Data connection protection flows

The following subcommands are available:

- ▶ Client issues PROTECT SAFE or SAFE subcommand

The protected buffer size is negotiated with the server using the PBSZ command. A PROT S command is sent to the server and the server replies. Any data subsequently transferred on the data connection will be integrity protected and sent in buffers no larger than the size negotiated on the PBSZ command.
- ▶ Client issues PROTECT PRIVATE or PRIVATE subcommand

The protected buffer size is negotiated with the server using the PBSZ command. A PROT P command is sent to the server and the server replies. Any data subsequently transferred on the data connection will be integrity protected, encrypted, and sent in buffers no larger than the size negotiated on the PBSZ command.
- ▶ Client issues PROTECT CLEAR or CLEAR subcommand

A PROT C command is sent to the server and the server replies. Any data subsequently sent on the data connection will be sent unprotected.

11.1.4 FTP and Kerberos scenario

We include a sample scenario here for completeness.

FTP attempt without Kerberos credentials

This scenario shows the error that is returned if you try to start an FTP session with a Kerberized FTP server and you do not have your initial credentials (obtained with the `knit` command).

Figure 11-42 on page 284 shows the FTP server responds with message EUVF02010E No security credential supplied when the client has not obtained credentials.

```

FOCAS @ SC63:/SC63/tmp>ftp wtsc63c
Using 7-bit conversion derived from 'ISO8859-1' and 'IBM-1047' for the control c
onnection.
Using the same translate tables for the control and data connections.
IBM FTP CS V1R2
FTP: using TCPIPC
Using catalog '/usr/lib/nls/msg/C/ftpdmsg.cat' for FTP messages.
Connecting to: wtsc63c.zos12.ral.ibm.com 9.12.6.61 port: 21.
220-FTPDC1 IBM FTP CS V1R2 at wtsc63c.zos12.ral.ibm.com, 12:06:10 on 2002-05-31.
220 Connection will close if idle for more than 5 minutes.
FC0155 ftpAuth: security values: mech=GSSAPI, sFTP=R, sCC=C, sDC=C
>>> AUTH GSSAPI
334 Using authentication mechanism GSSAPI
FC1429 gss_user_error: entered with major: 70000 minor: 25ea102
GSSAPI error major status code: 70000 - EUVF02010E No security credential suppli
ed
GSSAPI error minor status code: 25ea102 - Ticket-granting ticket is not found in
credentials cache
Request to initialize the security context failed
Authentication negotiation failed
Unable to successfully negotiate required authentication
>>> QUIT
221 Quit command received. Goodbye.
Command:
quit

```

Figure 11-42 Error when no credentials have been obtained before FTP attempt

To obtain the initial Kerberos credentials for a Kerberos principal, use the **kinit** command (either TSO or UNIX)

```

FOCAS @ SC63:/SC63/tmp>klist
Ticket cache: FILE:/var/skrb/creds/krbcred_cf635eb0
Default principal: FOCAS@ZOS12.RAL.IBM.COM
FOCAS @ SC63:/SC63/tmp>kinit FOCAS
EUVF06017R Enter password:

FOCAS @ SC63:/SC63/tmp>klist
Ticket cache: FILE:/var/skrb/creds/krbcred_cf768b40
Default principal: FOCAS@ZOS12.RAL.IBM.COM

Server: krbtgt/ZOS12.RAL.IBM.COM@ZOS12.RAL.IBM.COM
Valid 2002/05/31-08:12:36 to 2002/05/31-18:12:36
FOCAS @ SC63:/SC63/tmp>

```

Figure 11-43 Obtaining credentials with the **kinit** command

Figure 11-43 shows a user checking their credentials with the **klist** command. The first time **klist** is issued, Kerberos responds with the default principal name (obtained from the RACF user ID's KERB segment). The user then enters the **kinit** command to get a Kerberos principal's credentials. The **klist** command then shows that they were obtained correctly, and that the credentials are valid from 08:12 to 18:12.

FTP attempt without keytable entry

This scenario shows the error that is returned if you try to start an FTP session without previously defined keys in the keytable. Figure 11-44 shows the FTP server responding with the EUVF02010E message that it received from the KDC.

```
BMOS @ SC63: />ftp wtsc63
IBM FTP CS V1R2
FTP: using TCPIPC
Connecting to: wtsc63.zos12.ral.ibm.com 9.12.6.61 port: 21.
220-FTPDC1 IBM FTP CS V1R2 at wtsc63.zos12.ral.ibm.com, 20:08:56 on 2002-06-27.
220 Connection will close if idle for more than 5 minutes.
>>> AUTH GSSAPI
334 Using authentication mechanism GSSAPI
>>> ADAT YIICdQYJKoZIhvcSAQICAQBuggJkMIICYKADAgEFoQMCAQ6iBwMFACAAACjggGAYYIBfDC
CAXigAwIBBaETGxFaT1MxMi5SQUwuSUJNLkNPTaIqMCigAwIBA6EhMB8bA2Z0cBsYd3RzYzYzLnpczE
yLnJhbC5pYm0uY29to4IBLjCCASqgAwIBAAEDAgEGooIBHASCARizIeUjOJZ/K871Fv4g320+CyNhkEW
3JRhpFmKIV4RaCeorcr0A6ZmebxW+QdsLfaTgbm+rUCxD/UNRNn+XZck7rPp0uqfCwvTOXpvKvMIj9E1
4AGMHKrtsfAbrIs66PNYIY4VrpU5atKWyZ6rNu2wKFoCBQ5+DHM04r2DsOI fnRiXA87c5meyyOIf8oL f
7RtpLkZEmOBGfK3LzrNJF0mebZayg9zrJuEA9CFxmw4EvQrmhs0n4rqTfiYHMrARgqYno9Rno4xb+d8A
4ebvdWQ8GIcNck0DBtJ+cBW2ZiR/uhCvAOUNVutW5BQ8mfXKvWbGpKoMvpcGJneSdz5sq7/ewgPqcS2P
30LxQmtcHcz1l+0veBfBbh4P0pIHGMIHDoAMCAQGigbsEgbheOjTyjRbhT5VQ2dB4hmwzm6Y6xdrkz8+
A6J7WT/Ymcm6jfdNXdaq61A10UxuvKbW61z1wngaG6tFFok4+RTzfwYeb4nGkSwdGp5gzuhvGegpKaQ0
Wpa6Umzp9RRdoHpw8R7rRR8+Tt6nMYFbVoy0LAWQVgXio+BMPzEwDJJOZFj9/DQ3W3ZEM99rv6hjr0a
KgbrsXSHnVM0u6RmOnK0A0Ir0cjws3pVIg5zoUq0T1s1nWULb0Pd1
535-GSSAPI error major status code: 70000 - EUVF02010E No security credential su
plied
535-GSSAPI error minor status code: 25ea101 - Principal is not found in key tabl
e
535 Request to acquire security credentials failed
Authentication negotiation failed
NAME (wtsc63:BMOS):
```

Figure 11-44 Error when FTP server does not have the proper kerberos key

To obtain the Kerberos key for a Kerberos principal, first find out the key version being used for the FTP server from the RACF perspective. This must match what is in the Kerberos keytable. Use the TSO command LISTUSER as shown in Figure 11-45. The FTP started task runs under the RACF user TCPIPMVS. The information we are interested in is the KEY VERSION, which is 006.

```
lu tcpipmvs kerb noracf
-----
KERBNAME= ftp/wtsc63.zos12.ral.ibm.com
KEY VERSION= 006
KEY ENCRYPTION TYPE= DES DES3 DESD
***
```

Figure 11-45 RACF kerberos information for user tcpipmvs

Next, within the z/OS UNIX environment we need to verify if this information matches the existing values in the keytable (krb5.keytab). Enter the Kerberos **keytab list** command as shown in Figure 11-46 on page 286.

```
BMOS @ SC63:/SC63/etc/skrb>keytab list
Key table: /etc/skrb/krb5.keytab
```

Figure 11-46 Kerberos keytab list

Notice that there are no keys defined. At this point let us define a key for the FTP server by issuing the **keytab add** command followed by the **keytab list** command as shown in Figure 11-47.

```
BMOS @ SC63:/>keytab add ftp/wtsc63.zos12.ral.ibm.com -v 6
EUVF06048R Enter password:

EUVF06049R Re-enter password:

BMOS @ SC63:/>keytab list
Key table: /etc/skrb/krb5.keytab

Principal: ftp/wtsc63.zos12.ral.ibm.com@ZOS12.RAL.IBM.COM
Key version: 6
Key type: 56-bit DES
Entry timestamp: 2002/06/27-16:38:11

Principal: ftp/wtsc63.zos12.ral.ibm.com@ZOS12.RAL.IBM.COM
Key version: 6
Key type: 56-bit DES using key derivation
Entry timestamp: 2002/06/27-16:38:11

Principal: ftp/wtsc63.zos12.ral.ibm.com@ZOS12.RAL.IBM.COM
Key version: 6
Key type: 168-bit DES using key derivation
Entry timestamp: 2002/06/27-16:38:11
```

Figure 11-47 Adding Keytab entry for ftp/wtsc63.zos12.ral.ibm.com

Successful FTP connection using Kerberos

Figure 11-48 on page 287 illustrates a successful FTP connection using Kerberos. Notice the message Authentication negotiation succeeded. This follows the ADAT command which is the passing of the client's Kerberos credentials. The FTP user is now ready to log on with his user ID and password.

```

BMOS @ SC63: />ftp wtsc63
IBM FTP CS V1R2
FTP: using TCPIP
Connecting to: wtsc63.zos12.ra1.ibm.com 9.12.6.61 port: 21.
220-FTPDC1 IBM FTP CS V1R2 at wtsc63.zos12.ra1.ibm.com, 20:46:23 on 2002-06-27.
220 Connection will close if idle for more than 5 minutes.
>>> AUTH GSSAPI
334 Using authentication mechanism GSSAPI
>>> ADAT YIICdQYJKoZIhvcSAQICAQBuggJKMIICYKADAgEFoQMCAQ6iBwMFACAAAACjggGAYYIBfDC
CAXigAwIBBaETGxFaT1MxMi5SQUwuSUJNLKNPtaIqMCigAwIBA6EhMB8bA2Z0cBsYd3RzYzYzLnpvczE
yLnJhbC5pYmOuY29to4IBLjCCASqgAwIBAaEDAgEGooIBHASCARGZFTSeUpP+gITjIMG71GV8LzVbdB6
joU5dieW+DnuQca3rIIuRmh6YFX3LAof6aIToaDB4AM1ix9LG33Mz080zQP7QXq14hMJoFcZWI+QrUJF
KmIzDqnzGBirWpc7XnX52b7VQn3HcK0ftX+R/1dKS/45dTsVhoGISNmuUteQoqnGaRDY0u95yTLkkC1B
wI8bSvLnvUTRNEUk/WuEHprxLVVP1pbMaCcxhHBGbeh3S/77D8zuV1GW3Um6mZ490PZVG1UB1g4naNU
pp9DCCTKfXr/CUHUNRtET7MXeLzwjfmHziwZ/b/RkiLV0UGgBfk52kj3QU880LTNOCvAtAtYJx6rrnoJ
FdUZx779ydzwx1LKyGymf4s9pIHGMIHDoAMCAQGiGbsEgbiRzg6qS2arKsGCBRbESMnw2uxm2nHKvA1
mXIYFR127LARJZEMoB5pcOC2oeW8yE0cApEjnpHrRCI90bqRtZFYs1ZbnMOUESqorBVeZy1lPp7shnM
7yae64cKVBC8ohWCuFEc5n6WiUHKtzIP4sW+QzMDuFZ7SYJpdg50XSGowM1+6H4sewMPPVkyFW4eJjao
1gceVB9EecQSoQ5u0dJYbkzCE8msa34Vud8XKEzXtU0o8vkrkeEsn
235 ADAT=YGgGCSqGSIB3EgECAgIAb1kwV6ADAgEFoQMCAQ+iSzBjoAMCAQGiQgRAR0MeuLJdt20x/H4
qZfGfrWFR3LH2Ry6irDkeauk1HLal8QNA5dVQrAo3MZECEFIing+UeNpPjwx9yYzR/B/I8Pg==
Authentication negotiation succeeded
NAME (wtsc63:BMOS):

```

Figure 11-48 Successful FTP session using Kerberos

11.2 z/OS TFTP server

The TFTP protocol is a standard protocol with STD number 33. Its status is elective and it is described in RFC 1350. Updates to TFTP can be found in the following RFCs: 1785, 2347, 2348, and 2349.

TFTP is an extremely simple protocol to transfer files. It is implemented on top of the User Datagram Protocol (UDP). The TFTP client initially sends a read/write request via port 69, then the server and the client determine the port that they will use for the rest of the connection. TFTP lacks most of the features of FTP; the only thing it can do is read or write a file from or to a server.

TFTP is installed in the `/usr/lpp/tcpip/sbin/` directory.

To start the TFTP server from the command line, type the `tftpd` command as follows:

```
tftpd [-l] [-p port] [-t timeout] [-r maxretries] [-c concurrency_limit] [-s maxsegsz]
[-f file] [-a archive directory [-a ...]] [directory ...]
```

Note: The TFTP server uses well-known port 69. The TFTP server has *no user authentication*. Any client that can connect to port 69 on the server has access to TFTP. If the TFTP server is started without a home directory, it allows *access to the entire HFS*. To restrict access to the HFS, start the TFTP server with a list of directories.

If you want to use the TFTP server on z/OS, you should specify an absolute path name for a directory. You may specify no more than 20 directories in the `tftpd` command.

If a list of directories is specified, only those specified directories are active. That list is used as a search path for incoming requests that specify a relative path name for a file. Activating a directory activates all of its subdirectories. TFTP will not allow access to any other parts of the file system.

On the other hand, if the TFTP server is started without a list of directories, all mounted directories are considered active. In this case, all HFS files are open to every TFTP client because TFTP does not provide user authentication.

In addition, the TFTP server pre-forks a child process to handle incoming requests when the concurrency limit is exceeded. Consequently, immediately after starting the TFTP server, two TFTP processes exist.

In case of a flood of concurrent TFTP requests, the TFTP server may fork additional processes. When the number of concurrent requests being processed drops below the concurrency limit, the number of TFTP processes is decreased to two.

TFTP may sound like a very dangerous alternative to FTP, but it is extensively used to download code and initial configurations to routers and simple workstations, because of its simplicity. Because of its lack of security, you ought to take these simple steps if you need to run a TFTP server on your z/OS:

- ▶ Ensure that the TFTP home directory list is short and harmless.
- ▶ Use IPSec whenever a non-secure network is involved.
- ▶ Restrict access to port 69 on packet filters in firewalls.

11.3 z/OS NFS server

The Network File System (NFS) was developed by Sun Microsystems in 1985 and has been implemented on various platforms such as Sun Solaris, HP/UX, AIX, Linux, Windows, z/OS and so forth. NFS clients can use the resources of NFS servers transparently as if they were their own local file systems.

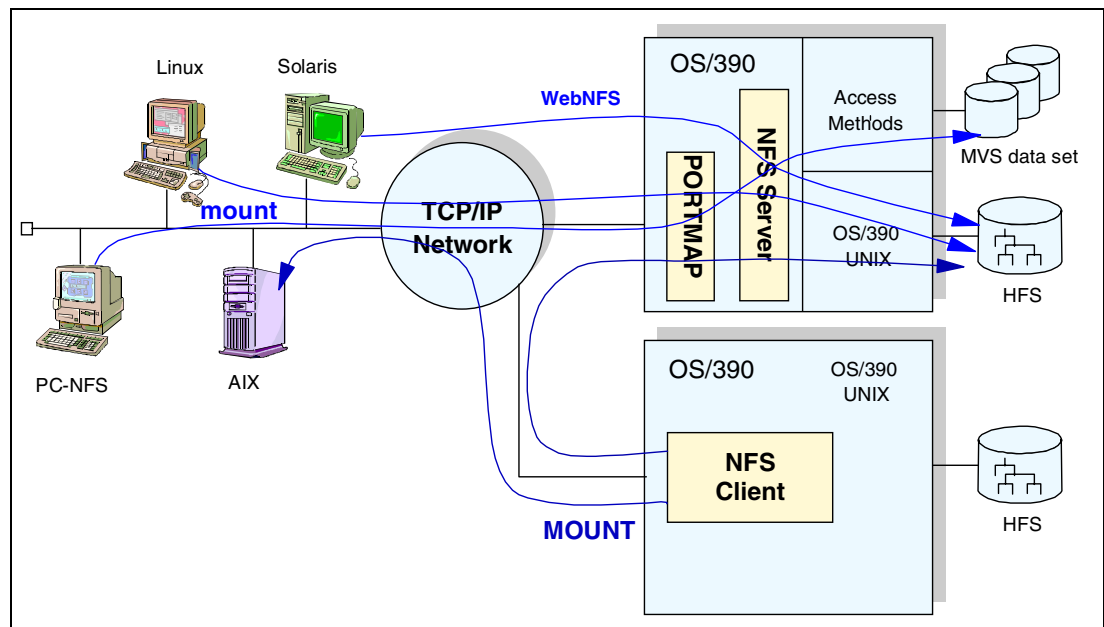


Figure 11-49 z/OS NFS overview

Client systems in a TCP/IP network that support the NFS client protocol can use traditional MVS data sets and UNIX System Services HFS files as part of their file system. Currently, z/OS UNIX System Services HFS files cannot be physically shared in read/write mode among multiple z/OS systems. Therefore, the combination of the z/OS NFS server with the z/OS NFS client is one of the alternative solutions that you can use to share z/OS UNIX System Services HFS files through a TCP/IP network.

NFS can use TCP or UDP to transport its data, and has been enhanced in recent releases of z/OS. Since the purpose of this section is to discuss z/OS NFS security, if you want to know more about the z/OS NFS server, see *z/OS Network File System Customization and Operation*, SC26-7253.

11.3.1 z/OS NFS security levels

The z/OS NFS server provides the following four security levels. These security levels are controlled by the security parameter in the z/OS NFS attributes data set:

- ▶ Unrestricted data access

When you specify `security(none)` in the attributes data set, no security checking is performed. We never recommend the use of this security level.

- ▶ Exports list checking

When you specify `security(exports)` in the attributes data set, the EXPORTS data set is used to check security. The EXPORTS data set is an MVS data set that is specified in the EXPORTS DD statement of the NFS server started procedure. The EXPORTS data set corresponds to the UNIX `/etc/exports` file.

The EXPORTS data set can control which client users can mount which MVS data sets, based on the client machine's specified IP address. It can also control which client users can mount all or part of the HFS. The entries in the EXPORTS data set specify which MVS high-level qualifiers or HFS directories can be mounted. The system administrator can use this data set to limit mounts to accredited clients only.

Note, however, that it is relatively easy to spoof an IP address, so exports list checking basically does not provide any real security.

- ▶ SAF checking

When you specify `security(saf)` in the attributes data set, System Authorization Facility (SAF) checking is performed. RACF, or an equivalent security product, provides the security information.

The server uses SAF to validate the z/OS user ID and password supplied by the client user. It also uses SAF to validate that the client user is allowed to access the data set on MVS. A RACF user ID must be defined for each client user that requires access to the server.

For HFS data, z/OS checks the UNIX permission bits before granting file access to a client user. The permission checking is based on the z/OS UNIX System Services UID obtained by the `mvlogin` process, not the UID passed in by the client on the RPC request. For users accessing HFS, their RACF user ID must have an OMVS segment defined in the RACF profile.

- ▶ Both SAF and exports list checking

When you specify `security(safexp)` in the attributes data set, NFS checks for both RACF authorization and exports list authorization before granting a client user access to z/OS data. For HFS data, z/OS checks the UNIX permission bits before granting file access to a client user. The UNIX permission checking is based on the z/OS UNIX System Services UID obtained by the `mvslogin` process, not the UID passed in by the client on the RPC request. This is the most restrictive means of limiting DASD access. It requires client users to use the `mvslogin` command.

Note: If you specified `security(saf)` or `security(safexp)` in your z/OS NFS attributes data set, you must download the `mvslogin` and `mvslogout` client commands to the NFS client system. For more information, see *z/OS Network File System Customization and Operation*, SC26-7253.

11.3.2 Security information exchange between NFS client and server

This section describes how an IP host name and a RACF user ID are provided to MVS NFS.

1. How is the client IP host name resolved by the NFS server?

If an NFS client connects to the NFS server, only the IP address is transmitted over the network. On the NFS server side, the client IP address is resolved into a client host name by using either local hosts tables or a query to a local or remote name server. It should be noted that in most implementations it is fairly easy to use a fake IP address on the client side. As mentioned previously, the `EXPORTS` data set simply lists the host name permitted to mount some directories of the host on which the NFS server is running. If a malicious user can get at IP address information listed in the `EXPORTS` data set by using the `showmount` command (see Figure 11-50) and tries to mount using this IP address illegally, unfortunately the NFS server permits this operation. Therefore, in the z/OS environment, it is better to use both SAF and `EXPORTS` list checking.

```
takada@rs600020[/home/takada]showmount -e mvs03a 1
export list for mvs03a:
/HFS/u/takada      rs60002 02 2
/HFS/etc           (everyone) 3
/TCPIP.TCPPARMS.R2505 mvs03c 4
```

Figure 11-50 Display exported directory information on z/OS

1 This command shows a list of the directories on z/OS host mvs03a that some NFS clients can mount.

2 This means that host rs600020 can mount the HFS directory /u/takada.

3 This means that every host can mount the HFS directory /etc.

4 `TCPIP.TCPPARMS.R2505` is a partitioned MVS data set. The host mvs03c can mount this data set as if this were a directory. PDS members are regarded as files.

2. How is a RACF user ID provided to NFS?

For single user PC systems, this is usually a simple task. Most PC systems support an extension to the NFS protocol called PC-NFS client. The z/OS NFS server has, like many NFS implementations, a PC-NFS server. PC-NFS support is enabled by specifying the `pcnfsd` verb in the NFS attributes data set. If PC-NFS support is enabled on the NFS server, after a `mount` command is received from the NFS client, the server prompts the client asking for a user ID and password. Both are checked by RACF.

In our test environment, we used the InterDrive NT NFS Client provided by FTP Software. Figure 11-51 shows how to specify the security information. In this case, you have to specify z/OS's host name or IP address in the Authentication server field and then fill in the RACF user ID and password in the Username and Password fields.

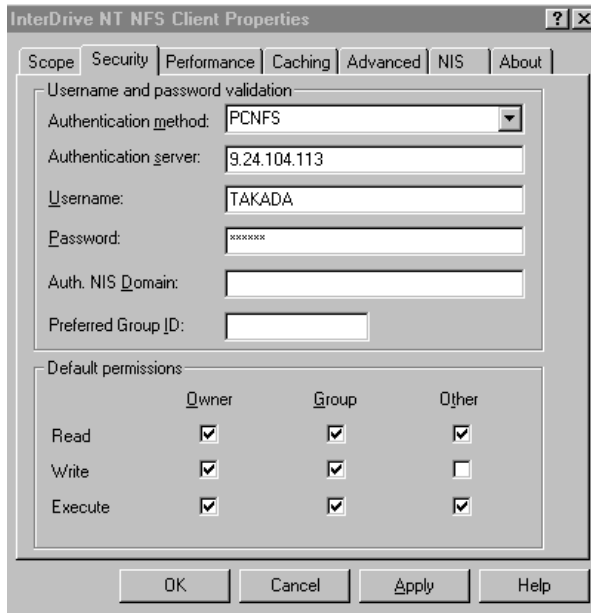


Figure 11-51 Defining RACF user ID and password in PC-NFS client configuration

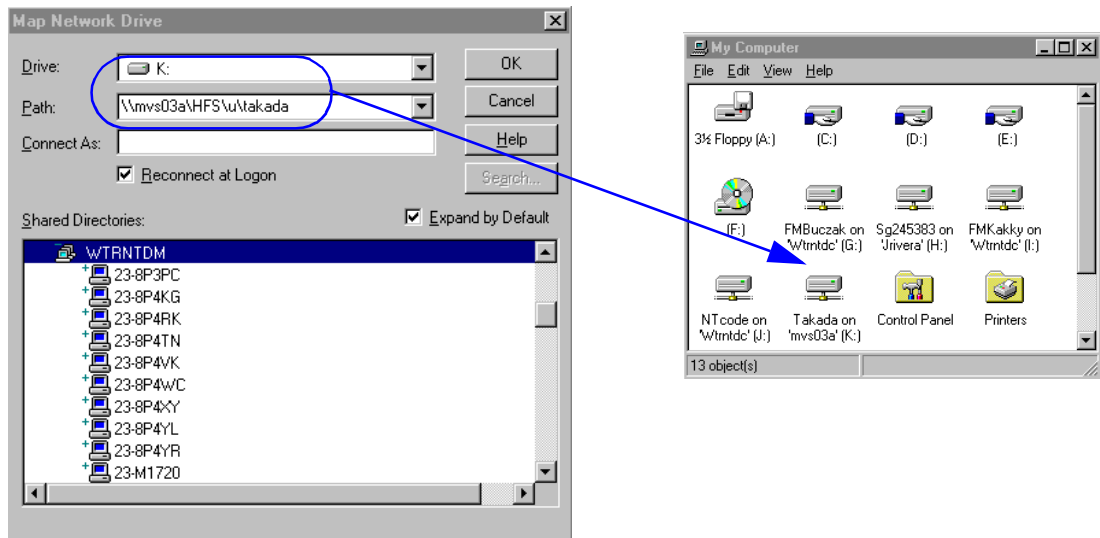


Figure 11-52 Mount processing on Windows NT client

When you want to mount z/OS resources (MVS data sets, HFS or both) in your Windows NT environment, you have to map the z/OS resources to logical drives. First, right-click the **Network Neighborhood** icon on the desktop and then select **Properties**. After the Map Network Drive window appears (Figure 11-52), fill in the z/OS resources in the Path

field and click **OK**. Meanwhile, the RACF user ID and password are sent to the z/OS security server specified in Figure 11-51 on page 291 and then the z/OS security server checks if they are valid. z/OS resources you specify in the Path field must be expressed in the following manner:

```
\\hostname\resource_name
```

In this case, `hostname` is `mvs03a` and `resource_name` is `\HFS\utakada`. Now you can use OS/390 resources as if they were local disk drives.

- ▶ A problem arises with UNIX systems, which usually do not provide a PC-NFS client, because they are by design multiuser systems.

In most cases on a UNIX system, users must have superuser authority (usually called root authority) to issue a `mount` command. Because standard mount processing does not provide user ID and password checking, NFS running on z/OS has implemented a protocol extension to mount called `mvsmount`. The `mvsmount` protocol extension is implemented by the commands `mvslogin` and `MImvslogout`, which allow the entry of a RACF user ID and password on several UNIX platforms.

To enable client users to access the MVS system, you must install the `mvslogin` and `mvslogout` commands on the client workstations. The source code to install these commands is provided in the `hlq.NFSTARB` data set. For some client machines, you might need to modify the code to port these commands so they run on your client machine.

In our environment, we ported these clients to the AIX V4.3.2 operating system. Figure 11-52 on page 291 shows how to use the `mvslogin` and `mvslogout` commands on the AIX operating system. In this case, we specified `security(safexp)` in the NFS attribute data set.

```

takada@rs600020[/home/takada]showmount -e mvs03a1
export list for mvs03a:
/HFS/u/takada      rs600020
/HFS/etc           (everyone)
/TCPIP.TCPPARMS.R2505 mvs03c
takada@rs600020[/home/takada]su2
root's Password:
takada@rs600020[/home/takada]mount mvs03a:/HFS/u/takada hfsdir3
takada@rs600020[/home/takada]mount mvs03a:/HFS/etc etcdir
takada@rs600020[/home/takada]mount mvs03a:/TCPIP.TCPPARMS.R2505 mvsdir
mount: access denied for mvs03a:/TCPIP.TCPPARMS.R2505
mount: giving up on:
      mvs03a:/TCPIP.TCPPARMS.R2505
Permission denied
takada@rs600020[/home/takada]mount4
node      mounted      mounted over   vfs      date      options
-----
      /dev/hd4      /              jfs      Aug 09 13:57 rw,log=/dev/hd8
      /dev/hd2      /usr           jfs      Aug 09 13:57 rw,log=/dev/hd8
      /dev/hd9var   /var          jfs      Aug 09 13:57 rw,log=/dev/hd8
      /dev/hd3      /tmp          jfs      Aug 09 13:57 rw,log=/dev/hd8
      /dev/hd1      /home         jfs      Aug 09 13:58 rw,log=/dev/hd8
      /dev/lvkakky  /home/kakky   jfs      Aug 09 13:58 rw,log=/dev/hd8
      /dev/lv00     /usr/inst.images jfs      Aug 09 16:40 rw,log=/dev/hd8
mvs03a    /HFS/u/takada  /home/takada/hfsdir nfs3     Aug 19 11:21

mvs03a    /HFS/etc       /home/takada/etcdir nfs3     Aug 19 11:21
takada@rs600020[/home/takada]exit
takada@rs600020[/home/takada]mvslogin mvs03a takada5
Password required
GFSA973A Enter MVS password:*****6
GFSA978I TAKADA logged in ok.7
:
:
takada@rs600020[/home/takada]mvslogout mvs03a3
GFSA958I uid 5002 logged out ok.

```

Figure 11-53 NFS mount process using mvslogin and mvslogout commands

- 1** You can check which directory you can mount by means of the **showmount -e** command.
- 2** To issue the **mount** command, you have to gain root authority.
- 3** This shows three mount processes. The reason why the last one fails is that hosts other than mvs03a cannot mount TCPIP.TCPPARMS.R2505 (1).
- 4** This command enables you to check the status of all mounted directories in your UNIX system.
- 5** This shows how to issue the **mvslogin** command; mvs03a is the host name of the NFS server and TAKADA is the RACF user ID.
- 6** You have to enter your password as registered in the z/OS RACF database.
- 7** The GFSA978I message means the authentication has succeeded. If the client UID and GID on the UNIX system do not match the UNIX UID and GID, you will see the following informational message that follows the GFSA978I message:

```

Mismatch in uid/gid: OpenEdition uid is 50011, gid is 3,
      client uid is 5002, gid is 1.

```

This informational message contains the z/OS UNIX UID and GID for the z/OS user identification.

3 This shows how to issue the `mvslogout` command.

Table 11-1 shows the z/OS NFS processing at the time of mount requests.

Table 11-1 z/OS NFS server processing of a mount request

Security Option	Export File	HFS file	MVS data set
none	Not required	No checking, exported	No checking, exported
saf	Not required	No checking, exported	No checking, exported
exports	Required	Checking export file	Checking export file
safexp	Required	Checking export file	Checking export file

11.3.3 Access to the HFS

HFS security is based on permission bits associated with an HFS file, UID and GID values associated with the file, and the requesting RACF user ID.

A UID associated with a user is a number specified in the OMVS segment of a RACF user ID. A GID associated with a user is a number specified in the OMVS segment of the default RACF group to which the user belongs.

Permission bits specify whether read, write, or execute permission is granted to the file owner, the group to which the file owner belongs, or to everyone. When a file is created, it is automatically associated with the UID of the user that creates the file (the file owner) and the GID of the directory it is in (the parent directory).

If a UNIX System Services user tries to access an HFS file, the UID and GID are compared with the UID and GID associated with the file. Depending on whether the values are equal, UNIX System Services grants the access rights of the file owner, the owner's group, or the rights that are granted to everyone.

If NFS is used to access HFS files, we must take into account which UIDs and GIDs are in effect on the NFS client system, and which security scheme is used with the z/OS NFS server.

Because the UID and GID are associated with an HFS file (and not related to a user name or group name), the `ls -l` command on an NFS client system will return different file-owning user names and file-owning group names to what is on the NFS server system, if the assignment of UIDs to user names and GIDs to group names is not consistent within the NFS network.

Also we have to consider which UID and GID is assigned to the NFS client users when they access an HFS file. This is dependent on the NFS security scheme that is in use:

- ▶ If EXPORTS security (`security(exports)`) is used, there is no identification to RACF. For this reason, the UID and GID values associated with the user ID acting on the NFS client system are used for UNIX System Services security checking. An exception to this rule is UID=0 (superuser). It is mapped to 65534. This prevents malicious users from accessing mounted OS/390 resources with superuser authority.
- ▶ If SAF security (`security(saf)` or `security(safexp)`) is used, the NFS client user has to identify itself to RACF by either a PC-NFS mount or the `mvslogin` command. RACF

associates UID and GID values with the NFS client user. These values are used in further processing on the z/OS NFS side. The problem is that additional security checking is done on the NFS client side. This complicates matters when the UID/GID values are not consistent in your network.

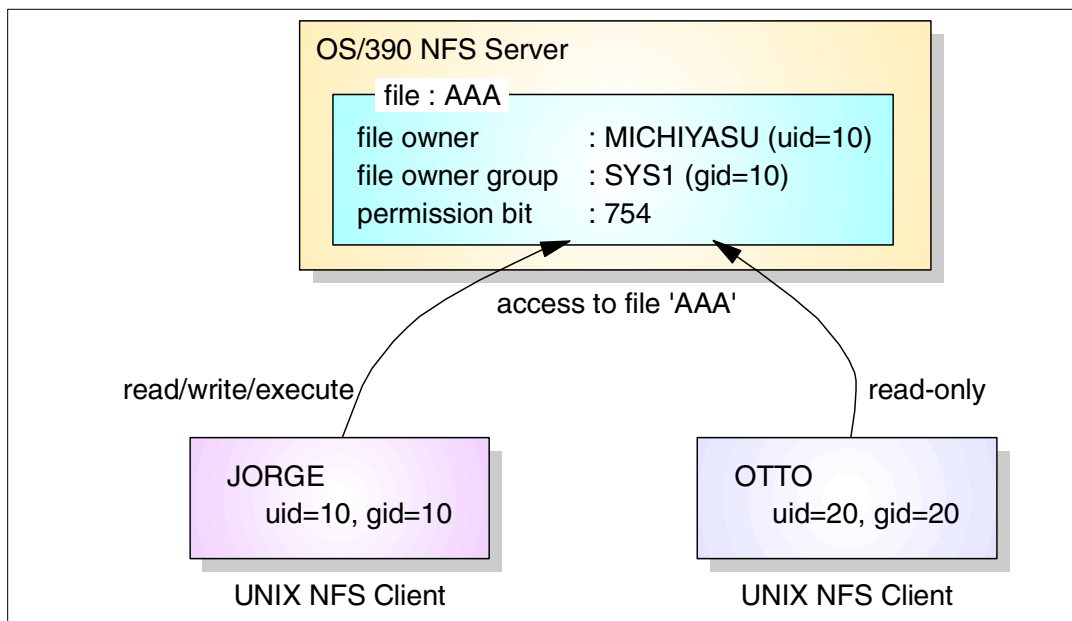


Figure 11-54 NFS connection between a UNIX NFS client and an MVS NFS server

Figure 11-54 shows an example of an NFS connection between a UNIX NFS client and a z/OS NFS server using EXPORTS security.

MICHiyASU is a user with UID=10, GID=10 defined under UNIX System Services and this user is an owner of the HFS file AAA. HFS file AAA has permission bits 754 which mean:

- ▶ The file owner can access this file with read/write/execute authority.
- ▶ Users who belong to the file owner group have read/execute authority.
- ▶ Other users can only read this file.

Now there are two other NFS client user IDs accessing the HFS file. Let us call them JORGE with UID=10, GID=10, and OTTO with UID=20, GID=20.

Note: Independent of the security scheme used, a user with superuser authority (UID=0) can issue NFS **mount** commands on UNIX systems.

JORGE will get access to MICHiyASU's files, that is, file AAA, as if it were MICHiyASU because they have the same UID and GID values (10). On the other hand, OTTO, which has a UID=20 and GID=20, gets world access, that is, read-only access.

If security(safexp) is specified in the z/OS NFS attribute data set, OTTO issues the **mvslogin** command and passes MICHiyASU's RACF user ID and password to the z/OS security server as shown in Figure 11-53 on page 293. OTTO can access the file AAA with READ/WRITE/EXECUTE authority.

Table 11-2 on page 296 lists how the z/OS NFS server handles a file request from an NFS client in each security option.

Table 11-2 z/OS NFS server processing of a file request

Security Option	mvslogin	HFS file	MVS data set
none	Not required	Check file permission bits	No checking
saf	Required	SAF check	SAF check
exports	Not required	Check file permission bits	No checking
safexp	Required	SAF check	SAF check

Note: MVS conventional data sets do not support UNIX permission bits in the file attribute structure. By disabling the SAF security, you lose authorization checking for file operation to MVS conventional data sets. Therefore, if you need to give NFS clients access to MVS data sets, you should enable SAF security and protect your MVS data sets by making a RACF MVS data set profile.

11.3.4 Conclusion

The following are our recommendations for using NFS with UNIX System Services:

1. Regardless of the security scheme you use, assign consistent UID and GID values in your NFS network. Each user should have the same UID and GID on every system he or she works on.
2. Enable the PC-NFS support in z/OS NFS and use PC-NFS where possible.
3. Use EXPORTS security when you can trust your UNIX system administration.
4. Use SAF security when your environment has additional security requirements.

With SAF security:

– Follow this sequence of commands when:

a. Mounting to z/OS NFS:

- i. Log in to the user ID root on the NFS client (if using a UNIX workstation).
- ii. Issue the **mvslogin** command. This is not required to mount, but it is useful to check whether everything is in order.
- iii. Issue the **mount** command.
- iv. Check access to the HFS directory by using the **df** command on UNIX or the **qmount** command on DOS and OS/2.

b. Using HFS:

- i. Log in to the user ID you want to work with in your UNIX environment.
- ii. Issue the **mvslogin** command.
- iii. You should now be able to access the mounted file system as permitted.
- iv. If you logged out from UNIX, issue the **mvslogin** command after the next UNIX login.

c. Unmounting from z/OS NFS:

- i. Log in to the user ID root on the NFS client (if using a UNIX workstation).
- ii. Be sure that no other user needs the mounted directory.
- iii. Issue the **umount** command.

- iv. Issue the **mvsllogout** command.
 - Tell the UNIX users which directories are mounted from z/OS NFS, and that they may have different access rights for HFS files than for local files if the UID and GID values do not match.
 - Because the EXPORTS file is not used, the NFS client's **showmount** command (**showexp** under OS/2) will receive the reply:
no exported file systems
 - Be aware that the user ID used for mvsllogin could also be used for other services such as rlogin, FTP, and Telnet.
5. SAFEXP security combines EXPORT and SAF security, making it the most secure NFS security level. However, because of its complexity (checking UID and GID values, along with user ID and passwords), this level of security may also cause additional confusion.
- Use SAFEXP security only if you want to hide parts of the HFS from the outside world.
 - Be aware that faking the IP host address is not a difficult task, especially on PC systems in an office environment.
 - Keep the EXPORTS file as simple as possible. You have more flexibility if you assign access rights to HFS files by using RACF user IDs with different UID and GID values for security checking.
 - With SAFEXP security, the **showmount** command will give a response, but this just reflects the EXPORTS file.

All other security exposures mentioned for SAF security apply also to SAFEXP security.



TN3270 security

The TN3270 server provides 3270 emulation for TN3270 clients, such as IBM's Host On-Demand (HOD) or Personal Communications (PCOMM). Securing TN3270 sessions is key to avoiding security exposures that exist on the Internet and most large-scale networks.

Encryption and authentication for TN3270 can be provided by using the Secure Sockets Layer (SSL) protocol. We cover the various configuration parameters available for SSL such as encryption schemes, client authentication levels, and key database location. See 12.1, "TN3270 SSL" on page 300.

The IETF TLS-based Telnet Security Draft is discussed, and how the TN3270 server supports negotiated security (although TN3270 can still only support SSL, not TLS). See 12.2, "Negotiated Telnet security" on page 312.

Authentication of clients is normally performed by the SNA applications being accessed through the TN3270 server. Through the use of SSL and a digital certificate at the client, the TN3270 server can authenticate the client, before they are given access to any SNA application. Additionally, when SSL client authentication is used in conjunction with the Express Logon Feature (ELF), a client can automatically log into an application without having to manually supply a user ID and password. See 12.3, "Express Logon Feature (ELF)" on page 326.

12.1 TN3270 SSL

The TN3270 server provides the ability to protect Telnet connections by using System SSL (described in 9.1, “Secure Sockets Layer (SSL)” on page 186). In an SSL session, all data is encrypted before it is sent to the client. Encrypted data received from the client is decrypted before the data is sent to VTAM. The flows between Telnet and VTAM are unchanged.

To use SSL, the TN3270 server must have a private key and access to an associated server certificate. The TN3270 server can obtain the server certificate and the keys from three different places:

- ▶ A KDB file stored in an MVS data set
- ▶ A KDB file stored in an HFS file
- ▶ The RACF database

SSL client authentication provides additional authentication and access control by checking client certificates at the server. This support prevents a client from obtaining a connection without an installation-approved certificate. There are three levels of client authentication, which are discussed in 12.1.2, “Client authentication” on page 302.

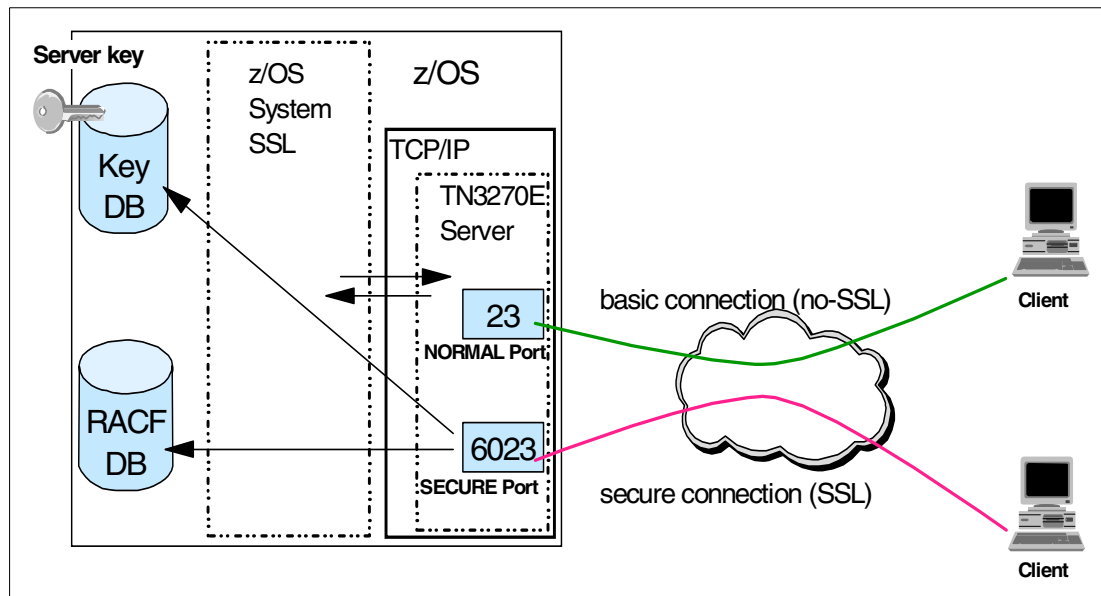


Figure 12-1 SSL protocol

Figure 12-1 shows how you might use different ports to separate a secure connection from a non-secure one. This is not necessary from the TN3270 server’s point of view, but it may be used to allow secure connections through a firewall, for example, if you did not want the well-known port 23 to be open to the public.

For a general discussion on digital certificates, see 2.2.4, “Digital certificates” on page 13 and for detailed information on how to obtain and manage certificates, including working examples for TN3270 use, see Chapter 10, “Certificate management in z/OS” on page 203.

12.1.1 TN3270 configuration parameters for SSL

There are two parameters that must be specified to implement SSL. The others are optional. The two essential parameters are:

► SECUREPORT

All SSL-enabled TN3270 ports must be defined by specifying a TELNETPARMS block for each port. The SECUREPORT port designation statement in the TELNETPARMS block indicates the port is capable of handling SSL connections.

► KEYSRING

The keyring type and location is specified in the KEYSRING statement. Only one keyring can be used by the TN3270 server.

The keyring can be defined in the TELNETGLOBALS or TELNETPARMS block. TELNETGLOBALS is the preferred definition method since it ensures that the same keyring has been defined for all SECUREPORTs.

Optional SSL parameters can only be specified for SECUREPORTs and can be specified in the TELNETGLOBALS, TELNETPARMS or PARMSGROUP blocks. Some of the optional parameters are detailed in 12.2.1, “TN3270 server parameters for negotiated security” on page 314 and in 12.3, “Express Logon Feature (ELF)” on page 326. They include the following:

► ENCRYPTION

The ENCRYPTION parameter is used to limit the encryption algorithms to only those included in the parameter statement. If this parameter is not specified, any encryption algorithm supported by the installed level of System SSL is available for use. Some reasons to use this parameter are:

- The applications supported on this port require a high level of security and the installation wants all data encrypted using a particular encryption algorithm.
- Certain connections are local and the installation does not require encryption for local clients. NULL encryption can be specified for this subset of connections.

► CONNTYPE

CONNTYPE parameter sets the level of security for connections. If CONNTYPE is not specified, a SECUREPORT defaults to CONNTYPE SECURE and a basic port defaults to CONNTYPE BASIC. Valid CONNTYPE options are:

- SECURE
- NEGTSURE
- BASIC
- ANY
- NONE

CONNTYPE parameter is detailed in “TN3270 server parameters for negotiated security” on page 314.

► CLIENTAUTH

The CLIENTAUTH parameter indicates that the client must send a client certificate to the server. If this parameter is not specified, a client certificate is not requested during the SSL handshake and no certificate-based client authentication is done. The different levels of client authentication are detailed in 12.1.2, “Client authentication” on page 302.

► CRLLDAPSERVER

The CRLLDAPSERVER parameter defines the name or IP address and port of the Certificate Revocation List (CRL) LDAP server. The CRL LDAP server is used (optionally) only if client authentication is configured. If CLIENTAUTH and the CRLLDAPSERVER have been specified, the certificate revocation list is checked during client authentication. If the client's certificate is found on the certificate revocation list, the connection is closed. Only one CRL LDAP server can be defined to the Telnet server.

Restriction: Currently the use of the Certificate Revocation List LDAP server is only supported for client certificates issued by Vault Registry. Vault Registry is an IBM product that provides the software necessary for an entity to become a Certificate Authority. Vault Registry is not an IBM Certificate Authority.

► SSLTIMEOUT

The SSLTIMEOUT parameter statement provides a unique timeout value for SSL handshake processing. This timeout limits the time SSL handshake processing waits for a client response.

This parameter is detailed in 12.2.1, “TN3270 server parameters for negotiated security” on page 314.

► EXPRESSLOGON / NOEXPRESSLOGON

The EXPRESSLOGON parameter statement allows a user at a workstation, with a TN3270 client and a X.509 certificate, to log on to an SNA application without entering a user ID or password. If this statement is not coded or NOEXPRESSLOGON is specified, EXPRESSLOGON function is not available to the client.

This parameter is detailed in 12.3, “Express Logon Feature (ELF)” on page 326.

12.1.2 Client authentication

There are three ascending levels of client authentication that can be defined for the TN3270 server.

► Level 1

Level 1 is the simplest level. To pass authentication, the Certificate Authority (CA) that signed the client certificate must be considered trusted by the server (that is, a certificate for the CA that issued the client certificate is listed as trusted in the server's keyring). If the client is using a self-signed certificate, then the client certificate needs to be in the server's keyring as a trusted CA certificate. You implement this level of authentication by coding CLIENTAUTH SSLCERT in the TCP/IP profile.

► Level 2

Level 2 authentication includes level 1 authentication. In addition, level 2 checking verifies that the client certificate has been registered with RACF (or other SAF-compliant security product that supports certificate registration). To implement level 2, the client certificate has to be added to the RACF database and associated with a RACF user ID. The TCP/IP configuration file must have a CLIENTAUTH SAFCERT statement.

► Level 3

This authentication level provides, in addition to level 1 and level 2 support, the capability to map access to the TN3270 server's port(s) with a SERVAUTH class profile in RACF. The RACF profile name for SERVAUTH class is:

EZB.TN3270.sysname.tcpname.PORTnnnnn

where nnnnn is the port number with leading zeros. For example, the profile name used in our ITSO environment is:

```
EZB.TN3270.SC64.TCPIPA.PORT23002
```

The profile name can contain wild cards to the extent that the security product allows. To protect all ports with a single profile, the following security product profile name could be used:

```
EZB.TN3270.SC64.TCPIPA.PORT*
```

The user ID returned by RACF for a client certificate is checked for READ access to the profile. If the SERVAUTH class is not active or the SERVAUTH profile for the server and port is not defined, it is assumed level 3 authentication is not needed and level 2 is assumed. Level 3 authentication is provided so that a client who has a certificate registered with RACF may be restricted to using particular ports on a particular TN3270 server.

To implement level 3 authentication you define the CLIENTAUTH SAFCERT statement in the TCP/IP profile (as for level 2) but you additionally define and activate a SERVAUTH class RACF profile for every port that you want to protect. Figure 12-2 shows an example of activating the SERVAUTH class (only needs to be done once) and defining a profile to protect port 23002 on TN3270 server TCPIPA on z/OS image SC64.

```
SETROPTS CLASSACT(SERVAUTH)
SETROPTS RACLIST(SERVAUTH)
RDEFINE SERVAUTH EZB.TN3270.SC64.TCPIPA.PORT23002 UACC(NONE)
PERMIT EZB.TN3270.SC64.TCPIPA.PORT23002 CL(SERVAUTH) ID(userid) ACCESS(READ)
SETROPTS RACLIST(SERVAUTH) REFRESH
```

Figure 12-2 Defining RACF SERVAUTH class and profile

In this example, userid is the RACF user ID associated to client certificate.

12.1.3 TN3270 server SSL configuration scenarios

This section illustrates some scenarios using the TN3270 SSL feature.

Using different ports for secure and non-secure TN3270

To use SSL, you configure the TN3270 Telnet server to use the RACF keyring (or key database in HFS) and the secure port. Figure 12-3 on page 304 shows the TN3270 SSL parameters defined in our example environment.

```

TELNETPARMS
  PORT 23
ENDTELNETPARMS

TELNETPARMS
  KEYRING SAF tcpipa.tn3270.keyring
  SECUREPORT 23001
  CONNTYPE SECURE
  CLIENTAUTH SSLCERT
ENDTELNETPARMS

BEGINVTAM
  PORT 23 23001
  DEFAULTLUS
    TCP65001 TCP64002 TCP64003 TCP64004 TCP64005
    TCP65006 TCP64007 TCP64008 TCP64009 TCP64010
    TCP65011 TCP64012 TCP64013 TCP64014 TCP64015
    TCP65016 TCP64017 TCP64018 TCP64019 TCP64020
    TCP65021 TCP64022 TCP64023 TCP64024 TCP64025
    TCP65026 TCP64027 TCP64028 TCP64029 TCP64030
  ENDDFAULTLUS
  ALLOWAPPL *
ENDVTAM

```

Figure 12-3 Sample TCP/IP profile

The KEYRING SAF tcpipa.tn3270.keyring statement specifies the name of the RACF keyring that the TN3270 server will use to look for the certificates it needs. In order to list the keyring and the certificates contained on the keyring, the server uses an internal RACDCERT LIST and RACDCERT LISTRING command, which are protected by RACF (see 10.4.1, “RACDCERT command security” on page 209). The RACF user ID associated with the TCP/IP started task therefore must be granted READ access to the appropriate RACF profiles, which are named IRR.DIGTCERT.LIST and IRR.DIGTCERT.LISTRING. The TCP/IP address space user ID in this example is TCPIPA.

```

PERMIT IRR.DIGTCERT.LIST CLASS(FACILITY) ID(TCPIPA) ACCESS(READ)
PERMIT IRR.DIGTCERT.LISTRING CLASS(FACILITY) ID(TCPIPA) ACCESS(READ)
SETR RACLIST(FACILITY) REFRESH

```

Figure 12-4 RACF commands to the TCPIP user ID

When you start the TCP/IP procedure, the messages shown in Figure 12-5 on page 305 will appear.


```

S TCPIPA
$HASP100 TCPIPA  ON STCINRDR
IEF695I START TCPIPA  WITH JOBNAME TCPIPA  IS ASSIGNED TO USER
TCPIPA  , GROUP SYS1
$HASP373 TCPIPA  STARTED
IEF403I TCPIPA - STARTED - TIME=10.26.50 - ASID=00FD.
IEE252I MEMBER CTIEZB00 FOUND IN SYS1.IBM.PARMLIB
IEE252I MEMBER CTIIDS00 FOUND IN SYS1.IBM.PARMLIB
EPW0250I EPWPITSK: FFST INITIALIZATION FOR TCP COMPLETE
EZZ0300I OPENED PROFILE FILE DD:PROFILE
EZZ0309I PROFILE PROCESSING BEGINNING FOR DD:PROFILE
EZZ0316I PROFILE PROCESSING COMPLETE FOR FILE DD:PROFILE
EZZ0334I IP FORWARDING IS DISABLED
EZZ0335I ICMP WILL NOT IGNORE REDIRECTS
EZZ0351I SOURCEVIPA SUPPORT IS ENABLED
EZZ0337I CLAWUSEDoublENOP IS CLEARED
EZZ0345I STOPONCLAWERROR IS DISABLED
EZZ0338I TCP PORTS 1 THRU 1023 ARE RESERVED
EZZ0338I UDP PORTS 1 THRU 1023 ARE RESERVED
EZZ4202I OPENEDITION-TCP/IP CONNECTION ESTABLISHED FOR TCPIPA
BPXF206I ROUTING INFORMATION FOR TRANSPORT DRIVER TCPIPA HAS BEEN
INITIALIZED OR UPDATED.
EZZ4313I INITIALIZATION COMPLETE FOR DEVICE OSA22E0
EZZB6473I TCP/IP STACK FUNCTIONS INITIALIZATION COMPLETE.
EZAIN11I ALL TCPIP INTERFACES FOR PROC TCPIPA ARE ACTIVE.
EZZ0400I TELNET/VTAM (SECOND PASS) BEGINNING FOR FILE: DD:PROFILE
EZZ6003I TELNET LISTENING ON PORT 23001
EZZ6003I TELNET LISTENING ON PORT 23
EZZ0403I TELNET/VTAM (SECOND PASS) COMPLETE FOR FILE: DD:PROFILE

```

Figure 12-5 TCP/IP startup

1 Be sure that the secure port was completely started and that TCP/IP is listening on that port.

You can check the port status with the D TCPIP,,T,PROF command illustrated in Figure 12-6.

```

D TCPIP,TCPIPA,T,PROF
EZZ6060I TELNET PROFILE DISPLAY 167
  PERS  FUNCT   DIA  SECURE  TIMERS  SMF  MAX  LINE
(LMTQ) (OATSSWH) (DRF) (SCKLECX) (IKPSTS) (ITIT) (RSQ) (BDCTT)
-----
---- --TS---- --- -B----- ---STS  ---- RSQ  --C--
----- PORT:    23  ACTIVE          PROF: CURR CONNS:    0
-----
---- --TS--- DJ-  SSS-DF- 1---STS  ---- RSQ  --C--
----- PORT:   23001  ACTIVE          PROF: CURR CONNS:    0
-----
KEYRING          : SAF tcpipa.tn3270.keyring 2
16 OF 16 RECORDS DISPLAYED

```

Figure 12-6 Display TCPIP profile

1 The flags in the SECURE column indicate which secure options are active for that port. The flag headings mean the following:

- S - secureport
- C - conntype. In the display above, port 23 has 'B' for basic and port 23001 has 'S' for secure.

- K - keyring
- L - crlldapserver
- E - encryption
- C - clientauth
- X - expresslogon

2 This is the keyring that will be used for all SSL connections. All of the secure ports will use the same keyring.

You can also check the port definitions and status with more details by using the D TCPIP,,T,PROF,PORT= command as shown in Figure 12-7 and Figure 12-8 on page 307.

```

D TCPIP,TCPIPA,T,PROF,PORT=23001,PROF=CURR,DET
EZZ6080I TELNET PROFILE DISPLAY 192
 PERS FUNCT  DIA  SECURE  TIMERS  SMF  MAX  LINE
(LMTQ) (OATSSWH) (DRF) (SCKLECX) (IKPSTS) (ITIT) (RSQ) (BDCTT)
----
---- --TS--- --- -B--D-- ---STS  ---- RSQ  --C-- *DEFAULT
---- ----- --- --S---- ----- ---- --- ----- *TGLOBAL
---- ----- DJ-  SS---F- ----- ---- --- ----- *TPARMS
---- --TS--- DJ-  SSS-DF- ---STS  ---- RSQ  --C--  CURR

PERSISTENCE
LUSESSIONPEND : OFF
MSG07          : OFF
TKOSPECLU(RECON): OFF
QUEUESESSION  : OFF

FUNCTIONS
OLDSOLICITOR  : OFF
SINGLEATTN     : OFF
TN3270E       : ON
SNAEXTENT     : ON
SIMCLIENTLU   : OFF
WLMCLUSTERNAME : OFF
HNLOOKUP      : OFF

DIAGNOSTICS
DEBUG          : DETAIL
DEBUG ROUTING  : JOBLG
FULLDATATRACE : OFF

SECURITY
SECUREPORT   : ON
CONNTYPE    : SECURE
KEYRING     : SAF tcpipa.tn3270.keyring
CRLLDAPSERVER : NONE
ENCRYPTION  : 4S,4M,3S,DS,4E,2E,NS,NM,NN DEF
CLIENTAUTH : SAFCERT
EXPRESSLOGON : OFF

TIMERS
INACTIVE       : OFF
KEEPINACTIVE   : OFF
PRTINACTIVE    : OFF
SCANINTERVAL   : 1800
TIMEMARK       : 10800
SSLTIMEOUT     : 5

```

Figure 12-7 PORT display (Part 1 of 2)

```

SMF
  SMFINIT      : OFF
  SMFTERM     : OFF
  SMFINIT TYPE119 : OFF
  SMFTERM TYPE119 : OFF
MAX LIMITS
  MAXRECEIVE   : 65536
  MAXVTAMSENDQ : 50
  MAXREQSESS   : 20
LINEMODE
  BINARYLINEMODE : OFF
  DISABLESGA     : OFF
  CODEPAGE  ASCII: IS08859-1   EBCDIC: IBM-1047
  DBCSTRANSFORM : OFF
  DBCSTRACE     : OFF
----- PORT: 23001 ACTIVE          PROF: CURR CONNS: 0
-----
53 OF 53 RECORDS DISPLAYED

```

Figure 12-8 PORT display (Part 2 of 2)

Using the same port for SSL and non-SSL TN3270

The previous example showed how you can configure the TN3270 server to use different ports for secure and non-secure traffic by having a TELNETPARMS block for each port. Separating the ports may be useful if you have a firewall between the TN3270 server and your clients, and one set of clients requires a secure connection (the public, for instance) and one set requires a non-secure connection (maybe your internal users).

You can use a single port to provide both secure and non-secure connections with the CONNTYPE ANY parameter. This can only be specified for a SECUREPORT.

Example 12-1 Sample setup of TN3270 server for SSL

```

PORT
  23 TCP INTCLIEN

TELNETGLOBALS
  KEYRING hfs /tn32v1r2.kdb          ;the HFS created by GSKKYMAN
ENDTELNETGLOBALS

TELNETPARMS
SECUREPORT 23
  CONNTYPE ANY
  CLIENTAUTH SSLCERT      ; client certificate must be issued by a trusted
ENDTELNETPARMS

BEGINVTAM
  PORT 23
  DEFAULTLUS
    TCP63031 TCP63032 TCP63033
  ENDDEFAULTLUS
  ALLOWAPPL *          ; Allow all applications that have not been
ENDVTAM
DEVICE OSA22E0 MPCIPA
LINK OSA22E0 IPAQNET OSA22E0

```

Example 12-1 on page 307 shows a sample TN3270 server configuration file. Key points to note are:

- ▶ The TELNETGLOBALS block has a KEYRING hfs /tn32v1r2.kdb statement. This specifies that the server's key database is contained in an HFS file named tn32v1r2.kdb in the root directory. This is where the server's certificate is stored, and since client authentication is used, a certificate for a CA that signed the client's certificate. Note that the keyring could equally have been in RACF if desired.
- ▶ The TELNETPARMS block has the statements
 - SECUREPORT 23. This specifies that port 23 may be used for an SSL connection from a client. Whether the port is used for SSL or not is influenced by the CONNTYPE parameter.
 - CONNTYPE ANY allows for a secure or non-secure (basic) connection through this port. A standard SSL handshake is attempted first. If the handshake times out, negotiated SSL is attempted. If that is rejected, a basic, non-secure connection is made.
 - CLIENTAUTH SSLCERT. This statement specifies that client authentication will be used, and that the only thing that needs to be checked on the client certificate is that we have a CA certificate for the issuer of the client certificate.

```
EZZ4202I OPENEDITION-TCP/IP CONNECTION ESTABLISHED FOR TCPIPC
EZZ4313I INITIALIZATION COMPLETE FOR DEVICE OSA22E0
EZB6473I TCP/IP STACK FUNCTIONS INITIALIZATION COMPLETE.
EZAIN11I ALL TCPIP INTERFACES FOR PROC TCPIPC ARE ACTIVE.
EZZ0400I TELNET/VTAM (SECOND PASS) BEGINNING FOR FILE: DD:PROFILE
EZZ6003I TELNET LISTENING ON PORT 23
EZZ0403I TELNET/VTAM (SECOND PASS) COMPLETE FOR FILE: DD:PROFILE
```

Figure 12-9 TCPIP startup with single port for both secure and non-secure connections

Figure 12-9 shows a portion of the system log from starting up the TCP/IP system configured in Example 12-1 on page 307. Note that there is only one listening port.

```
D TCPIP,TCPIPC,T,PROF
EZZ6060I TELNET PROFILE DISPLAY 183
PERS FUNCT DIA SECURE TIMERS SMF MAX LINE
(LMTQ) (OATSSWH) (DRF) (SCKLECX) (IKPSTS) (ITIT) (RSQ) (BDCTT)
-----
---- --TS--- --- SAH-DS- ---STS ---- RSQ --C--
----- PORT: 23 ACTIVE PROF: CURR CONNS:
-----
KEYRING : HFS /tn32v1r2.kdb
4 OF 4 RECORDS DISPLAYED
```

Figure 12-10 Display of TCP/IP profile with single CONNTYPE ANY port

Compare the display of the TCP/IP profile in Figure 12-10 with that of the stack that had separate ports for secure and non-secure connections in Figure 12-6 on page 305. The SECURE column CONNTYPE field (the 'C' in the SCKLECX heading) has an 'A', which means CONNTYPE ANY. In Figure 12-6 on page 305, port 23 had 'B' for basic and port 23001 had 'S' for secure.

```

D TCPIP,TCPIPC,T,CONN,PORT=ALL
EZZ6064I TELNET CONNECTION DISPLAY 207
      EN
CONN  TY IPADDR..PORT          LUNAME  APPLID  TSP  LOGMODE
-----
0000003C  9.24.104.119..2548      TCP63032          TPE
00000016 DS 9.24.104.119..2537      TCP63031          TPE
----- PORT:      23  ACTIVE          PROF: CURR CONNS:      2
-----
4 OF 4 RECORDS DISPLAYED

```

Figure 12-11 Command to display TN3270 connection on any port

As an example of how it is possible to connect to the same port (port 23 in this case) with no security specified at the client, and also with security, we started up two PCOMM TN3270 clients on a workstation. One client was configured for SSL, the other was not. Figure 12-11 shows the display of TN3270 connections with the D TCPIP,,T,CONN,PORT=ALL command. The second column (ENTY or ENcryption TYpe) shows a blank field for connection 3C and 'DS' (meaning DES encryption) for connection 16. The detail displays for connections 3C and 16 follow.

```

D TCPIP,TCPIPC,T,CONN,CONN=3C
EZZ6065I TELNET CONNECTION DISPLAY 374
CONN: 0000003C          CLNTIP..PORT: 9.24.104.119..2548
LINKNAME: VIPL090C063E  DESTIP..PORT: 9.12.6.62..23
HOSTNAME: NO HOSTNAME
CONNECTED: 15:22:31 05/17/2002 STATUS: SESSION PENDING
PORT: 23 ACTIVE SECURE ACCESS: NON-SECURE
PROTOCOL: TN3270E LOGMODE:          DEVICETYPE: IBM-3278-4-E
  OPTIONS: ETET--- 3270E FUNCTIONS: BSR----
                NEWENV FUNCTIONS: --
USERIDS
  RESTRICTAPPL: **N/A** CLIENTAUTH: **N/A**
  EXPRESSLOGON: **N/A**
APPL: **N/A**
LUNAME: TCP63032 TYPE: TERMINAL GENERIC
MAPS CONN IDENTIFIER OBJECT DEFAPPL          OPTIONS
LU MAPPINGS:
                >*DEFLUS* **N/A**          -----
DEFAULTAPPL: **N/A**
USS TABLE: **N/A**
INT TABLE: **N/A**
PARMS:
PERS FUNCT DIA SECURE TIMERS SMF MAX LINE
(LMTQ) (OATSSWH) (DRF) (SCKLECX) (IKPSTS) (ITIT) (RSQ) (BDCTT)
----
---- --TS--- --- -B--D-- ---STS ---- RSQ --C-- *DEFAULT
---- ----- --- --H---- ----- ---- --- ----- *TGLOBAL
---- ----- --- SA---S- ----- ---- --- ----- *TPARMS
---- --TS--- --- SAH-DS- ---STS ---- RSQ --C-- TP-CURR
---- --TS--- --- SAH-DS- ---STS ---- RSQ --C-- FINAL
28 OF 28 RECORDS DISPLAYED

```

Figure 12-12 Detail display of connection 3C showing non-secure access

Figure 12-12 shows the detail display of the non-secure connection 3C. ACCESS: NON-SECURE indicates that SSL is not being used on this connection, as expected.

```

D TCPIP,TCPIPC,T,CONN,CONN=16
EZZ6065I TELNET CONNECTION DISPLAY 382
CONN: 0000016          CLNTIP..PORT: 9.24.104.119..2537
LINKNAME: VIPL090C063E  DESTIP..PORT: 9.12.6.62..23
HOSTNAME: NO HOSTNAME
CONNECTED: 15:10:23 05/17/2002 STATUS: SESSION PENDING
PORT: 23 ACTIVE SECURE ACCESS: SECURE DS SSLCERT
PROTOCOL: TN3270E LOGMODE: DEVICETYPE: IBM-3278-4-E
  OPTIONS: ETET--- 3270E FUNCTIONS: BSR----
              NEWENV FUNCTIONS: --
USERIDS
  RESTRICTAPPL: **N/A** CLIENTAUTH: **N/A**
  EXPRESSLOGON: **N/A**
APPL: **N/A**
LUNAME: TCP63031 TYPE: TERMINAL GENERIC
MAPS CONN IDENTIFIER OBJECT DEFAPPL OPTIONS
LU MAPPINGS:
              >*DEFPLUS* **N/A**      -----
  DEFAULTAPPL: **N/A**
  USS TABLE: **N/A**
  INT TABLE: **N/A**
  PARMS:
PERS FUNCT DIA SECURE TIMERS SMF MAX LINE
(LMTQ) (OATSSWH) (DRF) (SCKLECX) (IKPSTS) (ITIT) (RSQ) (BDCTT)
----
---- --TS--- --- -B--D-- ---STS ---- RSQ --C-- *DEFAULT
---- ----- --- --H---- ----- ---- --- ----- *TGLOBAL
---- ----- --- SA---S- ----- ---- --- ----- *TPARMS
---- --TS--- --- SAH-DS- ---STS ---- RSQ --C-- TP-CURR
---- --TS--- --- SAH-DS- ---STS ---- RSQ --C-- FINAL
28 OF 28 RECORDS DISPLAYED

```

Figure 12-13 Detail display of connection 16 showing secure access

Figure 12-13 shows the detail display of the secure connection 16. ACCESS: SECURE DS SSLCERT indicates that SSL is being used, that the encryption algorithm is DES and that client authentication level 1 is being used (SSLCERT). Note the CLIENTAUTH: **N/A** field. As client authentication level 1 is being used (the only level available if you don't have the server's keyring in RACF), no RACF user ID is associated with the client certificate.

Configuring client authentication for the TN3270 server

The steps to configure the TN3270 server for SSL have been covered in the previous two scenarios. To implement client authentication you add the CLIENTAUTH statement to the TELNETPARM block for the port(s) where you want client authentication enabled. We cover authentication level 2 here (level 1 just requires that the CA that signed the certificate be present in the server's keyring, and level 3 is the same as level 2 except you have added one or more SERVAUTH class profiles in RACF to protect certain ports).

The basic steps to implement client authentication are:

- ▶ Obtain and store the server certificate as explained previously
- ▶ Implement client authentication:
 - Get a client certificate, either self-signed, generated by RACF, or an external CA.
 - Ensure the client certificate is in the client's key database.

- Add the client’s CA certificate into the server’s keyring as a CERTAUTH certificate. This ensures the client certificate, when received, can be validated. For a self-signed certificate, this CA certificate is the same as the client certificate. For an external CA-issued certificate, this is the certificate of the issuer. For a RACF-generated certificate, this is the certificate of the CA in the SIGNWITH parameter of the GENCERT command.
- Add the client certificate into the RACF database in “TRUST” status and associate it with a user.
- Test the connection

In the following example, we have made the following assumptions:

- ▶ The client is an IBM Personal Communications V5.0 client for Windows.
- ▶ A trial client certificate was obtained from an external CA company, Thawte.
- ▶ The server’s configuration is as shown in Example 12-2. This is a fragment of the configuration to show the appropriate SSL statements.

Example 12-2 Server configuration for client authentication scenario

```

PORT
  23 TCP INTCLIEN BIND 9.12.6.62

TELNETGLOBALS
  KEYRING SAF TN3270Ring           ;the RACF Key Ring Name
ENDTELNETGLOBALS

TELNETPARMS
  SECUREPORT 23
  DEBUG DETAIL CONSOLE
  CONNTYPE SECURE
  CLIENTAUTH SAFCERT       ; validate client cert through RACF for match
ENDTELNETPARMS

BEGINVTAM
  PORT 23
  DEFAULTLUS
    TCP63031 TCP63032 TCP63033
  ENDDEFAULTLUS
  ALLOWAPPL *           ; Allow all applications that have not been
ENDVTAM

```

For an overview of how the client certificate was obtained and stored, see “External CA-signed client certificate RACDCERT procedure” on page 241.

After the server and client certificates are in place, you must ensure the TN3270 client is configured to enable client authentication. The PCOMM client’s configuration of SSL can be reached by clicking **Communication -> Configure** then clicking the **Link Parameters** button. Select the **Enable Security** checkbox and the **Advanced Security** tab.

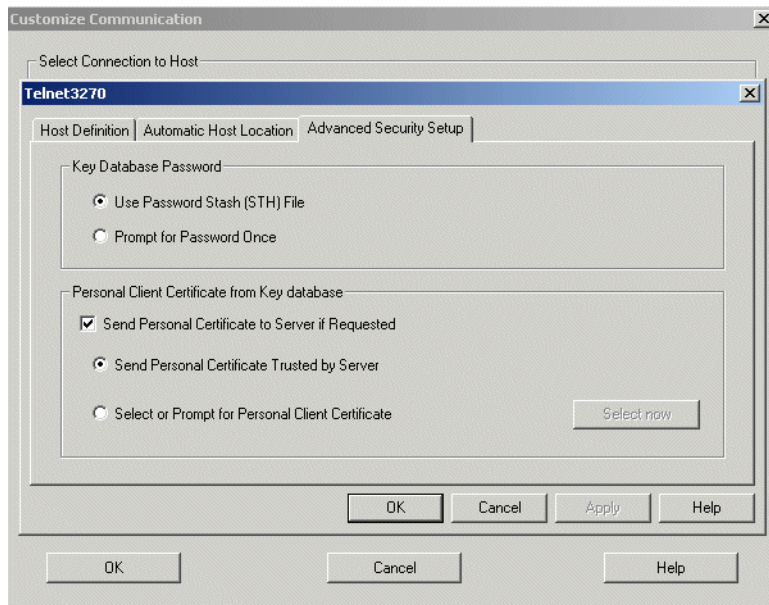


Figure 12-14 PCOMM client setup window for SSL security

Figure 12-14 shows the PCOMM client SSL security window. The Key Database part can be ignored, since it is used only if the server certificate is stored in an HFS file (in which case you must check **Use password stash (STH) file**). Select the **Send Personal Certificate To Server If Requested** check box and the **Send Personal Certificate Trusted by Server** radio button. That completes the client configuration.

12.2 Negotiated Telnet security

Negotiated Telnet is a name used in the z/OS IP manuals. In fact, negotiated Telnet is an implementation of the IETF TLS-based Telnet Security draft (see <http://www.ietf.org/proceedings/99mar/I-D/draft-ietf-tn3270e-telnet-tls-01.txt>). The name TLS-based Telnet Security is a little misleading because this IETF internet draft functions equally well with TLS 1.0 and SSL 3.0. In other words, the actual security protocol used with TLS-based Telnet Security could be SSL 3.0, and for TN3270, it is.

Note: At the time of writing, TLS (RFC 2246) is not supported on the TN3270 server. Only IETF Internet-Draft TLS-Based Telnet Security (negotiated Telnet) is supported.

What does the TLS-based Telnet Security define? It adds a new IAC (Interpret As Command) option and suboption. The START_TLS option allows the client (WILL START_TLS) or the server (DO START_TLS) to initiate a request for a secure session. Once TLS has been agreed upon, the session immediately drops into negotiation of either SSL or TLS. Negotiation of other Telnet IAC options is suspended until the security negotiation has successfully completed.

A typical TLS-base Telnet SSL flow would be:

1. IP connection establishment.
2. Telnet server sends the IAC DO START_TLS command to the client to verify if it wants to perform the SSL negotiation.
3. If a positive response is received, then Telnet begins a normal SSL handshake.

4. If no positive response is received, the connection is dropped.

Figure 12-15 shows the TLS flow.

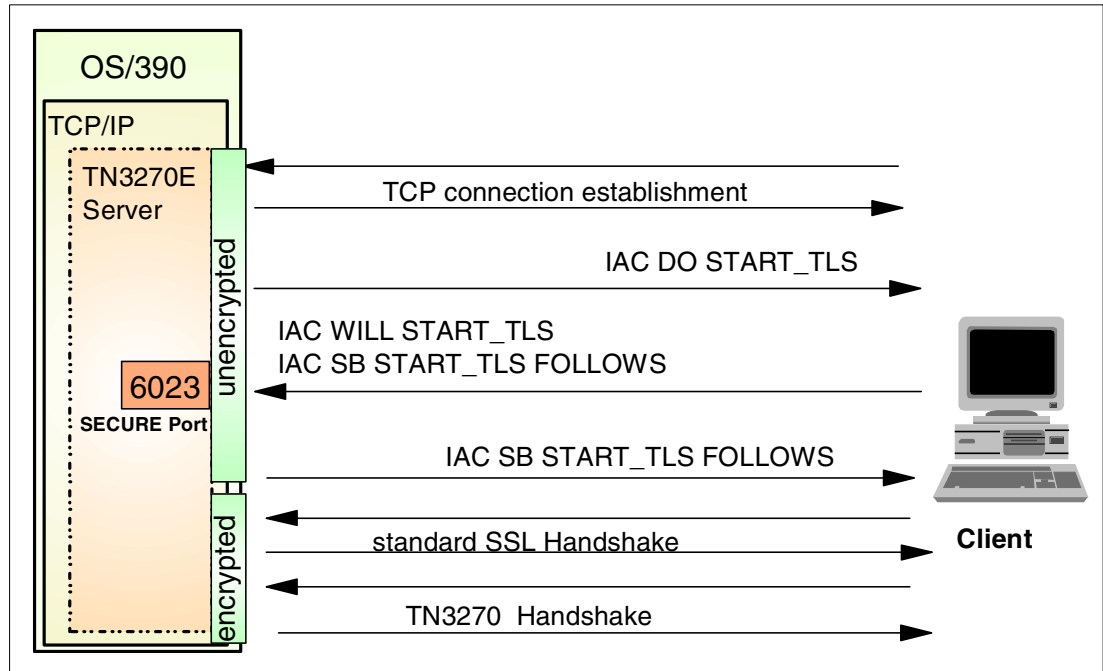


Figure 12-15 TLS-based telnet SSL flow

If the client and server cannot agree upon the START_TLS option, then the Telnet server can opt to drop into native TLS/SSL security negotiation (CONNTYPE SECURE in the TCP/IP profile data set).

Why add a new IAC option? The foremost advantage is that this option places the control of session security into the TN3270 world (instead of leaving it up to the transport layer). If a Telnet client won't accept a DO START_TLS option, the Telnet server can choose to end the session (CONNTYPE NEGTSURE in the TCP/IP profile data set).

The other significant advantage of placing encryption negotiation into the TN3270 option data stream is that a single port can be used for encrypted and non-encrypted sessions. Prior to negotiated Telnet, a separate port for secure and non-secure sessions had to be used. Since all TN3270 clients default to port 23, this was not an ideal situation.

Other advantages include being able to use session states to determine the status of a session. In addition, placing session security into the domain of TN3270 allows the new AUTH and ENCRYPT Telnet options to be used (if required). Both of these options (RFC 2941, *Telnet Authentication Option*, and RFC 2946, *Telnet Data Encryption Option*, respectively) have not been implemented in z/OS. However, they do indicate that the value of negotiated Telnet is also in its future capabilities.

12.2.1 TN3270 server parameters for negotiated security

TLS protocol allows the client and the server to negotiate the Telnet session before starting the connection. There are two parameters in TCP/IP profile related to TLS: CONNTYPE and SSLTIMEOUT:

► CONNTYPE

CONNTYPE parameter sets the level of security for connections. If CONNTYPE is not specified, a SECUREPORT defaults to CONNTYPE SECURE and a basic PORT defaults to CONNTYPE BASIC. Valid CONNTYPE options are:

- SECURE: Indicates that the SSL handshake will be used to start the SSL connection. If the client does not start the handshake within the time specified by SSLTIMEOUT, an attempt will be made to do a negotiated SSL handshake (as defined by the IETF TLS-based Telnet Security Draft). If the client rejects SSL, the connection will be closed.
- NEGOTSECURE: Indicates the client supports the IETF TLS-based Telnet Security Draft. A TN3270 negotiation with the client first determines if the client is willing to enter into a secure connection. If the client agrees, an SSL handshake is started and SSL protocols will be used for all subsequent communication. If the client rejects SSL, the connection will be closed.

If you know that the TN3270 SSL clients connecting into the port are using the protocol defined by the TLS-based Telnet Security Draft, you should consider using this option. With this option the SSL handshake is not attempted until after a positive response to the TN3270 DO STARTTLS IAC is received. This avoids the timeout delay that can occur when an SSL handshake is immediately started (as done with CONNTYPE SECURE) but the client is expecting the protocol used by the TLS-based Telnet Security Draft.

- BASIC: Indicates that a basic (non-SSL) connection will be used.
- ANY: Indicates that the connection can be either secure or basic. The TN3270 server will first try a standard SSL handshake. If the handshake times out, a negotiated SSL (see CONNTYPE NEGOTSECURE) is attempted.

If the client is willing to enter into a secure connection, SSL protocols will be used for all subsequent communication.

If the client is not willing to enter into a secure connection, a basic (non-SSL) connection is used.

- NONE: Indicates that no connection is allowed and the connection will be closed. If this option is specified in TELNETPARMS, a PARMSMAP must cover every allowable connection and the related PARMSGROUP must specify the desired CONNTYPE.

► SSLTIMEOUT

SSLTIMEOUT indicates how long Telnet will wait for an SSL handshake response before the request is dropped.

SSLTIMEOUT is different from the other timers. Telnet does not run this timer. The time value is passed to the SSL handshake process. If SSL does not get a response from the client within the SSLTIMEOUT period of time, the handshake request fails. Telnet will then proceed to the next available connection negotiation method or drop the connection.

When the connection is dropped, If DEBUG DETAIL is specified in TCP/IP profile, message EZZ6035I with RCODE 600E (SSL handshake timed out) is displayed.

12.2.2 TN3270 server configuration scenario

To use negotiated-TLS security, you customize the TN3270 Telnet server to use the CONNTYPE and SSLTIMEOUT parameters, besides the keyring and the secure port. In our ITSO environment, the parameters shown in Figure 12-16 were defined for the TN3270 server.

```
TELNETPARMS
KEYRING SAF tcpipa.tn3270.keyring
SECUREPORT 23001
CONNTYPE NEGTSURE
CLIENTAUTH SAFCERT
SSLTIMEOUT 5
DEBUG DETAIL
ENDTELNETPARMS
```

Figure 12-16 TLS TCP/IP profile definition

After starting TCP/IP, if you display port 23001, you can see the NEGTSURE and SSLTIMEOUT definition, as shown in Figure 12-17 on page 316 and Figure 12-18 on page 317.

```

D TCP/IP, TCP/IPA, T, PROF, PORT=23001, PROF=CURR, DET
EZZ6080I TELNET PROFILE DISPLAY 567
PERS  FUNCT   DIA  SECURE  TIMERS  SMF  MAX  LINE
(LMTQ) (OATSSWH) (DRF) (SCKLECX) (IKPSTS) (ITIT) (RSQ) (BDCTT)
----  -
----  --TS---  ---  -B--D--  ---STS  ----  RSQ  --C--  *DEFAULT
----  -
----  -S-----  -
----  -
----  -
----  DJ-  SN---F-  -----S  ----  -
----  --TS---  DJ-  SNS-DF-  ---STS  ----  RSQ  --C--  CURR

PERSISTENCE
LUSESSIONPEND : OFF
MSG07         : OFF
TKOSPECLU(RECON): OFF
QUEUESESSION  : OFF

FUNCTIONS
OLDSOLICITOR  : OFF
SINGLEATTN     : OFF
TN3270E       : ON
SNAEXTENT     : ON
SIMCLIENTLU   : OFF
WLMCLUSTERNAME : OFF
HNLOOKUP      : OFF

DIAGNOSTICS
DEBUG         : DETAIL
DEBUG ROUTING : JOBLG
FULLDATATRACE : OFF

SECURITY
SECUREPORT    : ON
CONNTYPE     : NEGTCURE
KEYRING       : SAF tcpipa.tn3270.keyring
CRLLDAPSERVER : NONE
ENCRYPTION    : 4S,4M,3S,DS,4E,2E,NS,NM,NN DEF
CLIENTAUTH   : SAFCERT
EXPRESSLOGON  : OFF

TIMERS
INACTIVE      : OFF
KEEPINACTIVE  : OFF
PRTINACTIVE   : OFF
SCANINTERVAL  : 1800
TIMEMARK      : 10800
SSLTIMEOUT   : 5

```

Figure 12-17 Display profile (Part 1 of 2)

```
SMF
  SMFINIT      : OFF
  SMFTERM     : OFF
  SMFINIT TYPE119 : OFF
  SMFTERM TYPE119 : OFF
MAX LIMITS
  MAXRECEIVE   : 65536
  MAXVTAMSENDQ : 50
  MAXREQSESS   : 20
LINEMODE
  BINARYLINEMODE : OFF
  DISABLESGA     : OFF
  CODEPAGE  ASCII: IS08859-1   EBCDIC: IBM-1047
  DBCSTRANSFORM : OFF
  DBCSTRACE     : OFF
----- PORT: 23001  ACTIVE          PROF: CURR CONNS: 0
-----
53 OF 53 RECORDS DISPLAYED
```

Figure 12-18 Display profile (Part 2 of 2)

12.2.3 TN3270 client (HOD) negotiated Telnet configuration scenario

In our ITSO environment we used Host On-Demand V6.0 to connect to TN3270 server.

TN3270 client (HOD) SSL configuration

To customize a HOD session for SSL and negotiated-Telnet, you can follow these steps:

1. On HOD, create a new session, such as that shown in Figure 12-19 on page 318. In this window, specify the Destination Address and Destination Port defined in TCP/IP profile. Then click the **Security** tab.

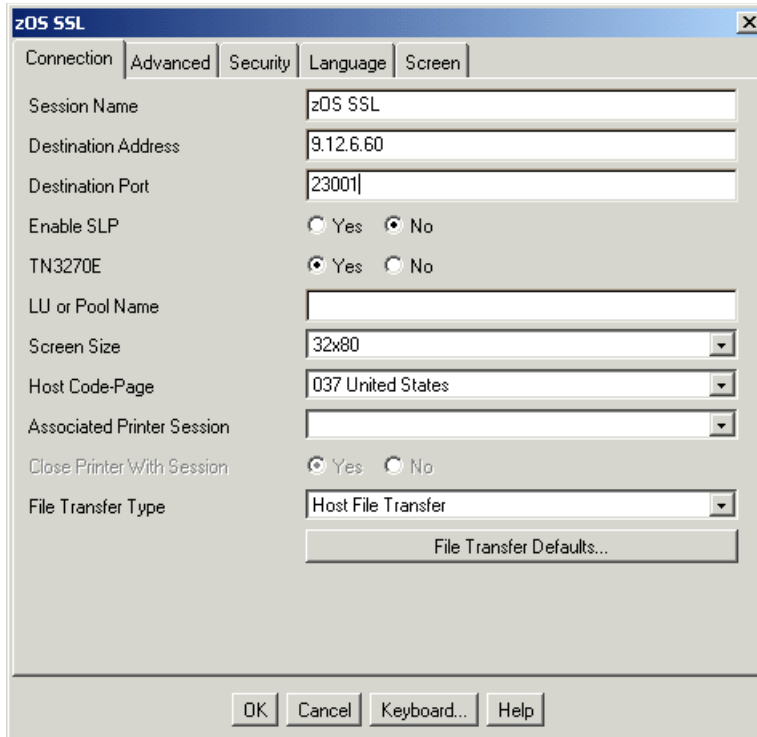


Figure 12-19 HOD session definition

2. On the Security tab, select Yes for **Enable Security (SSL)** and **Send a Certificate** (if you use client authentication).

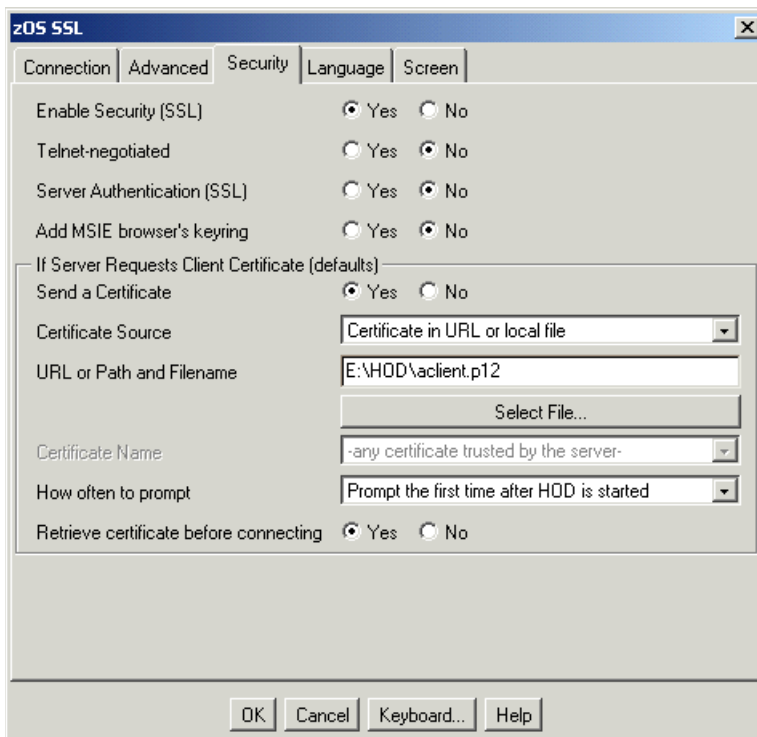


Figure 12-20 HOD Security definition

If you want to have Server Authentication, the Common Name (CN) defined for the server certificate must be the same as TCP/IP hostname.

3. If you choose **Send a certificate**, click **Select file** to specify the file name, location, and password of client certificate. The password is the same as used in the RACF RACDCERT EXPORT command. Then click **OK** in all of the windows.

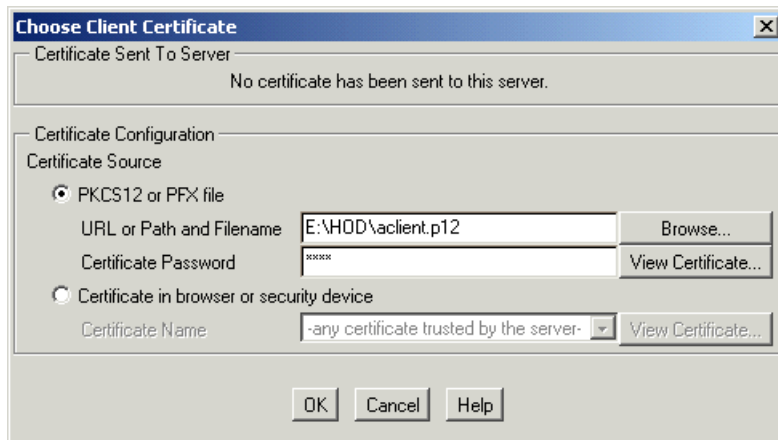


Figure 12-21 HOD client certificate definition

4. Now select the session you have created by double-clicking the session icon, as shown in Figure 12-22 on page 320.

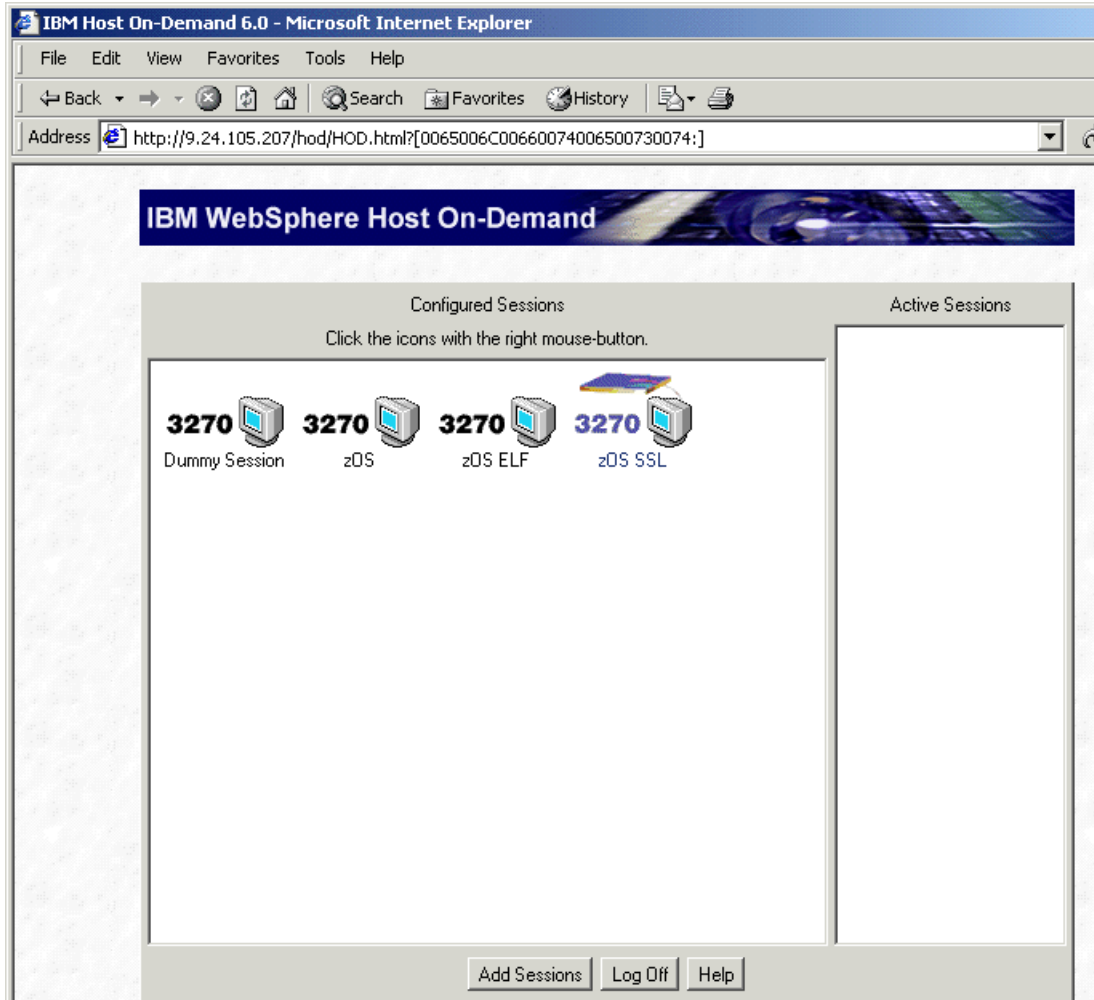


Figure 12-22 HOD starting the SSL session

5. The window shown in Figure 12-23 on page 321 will display. Supply the password to access PKCS12 file (aclient.p12).

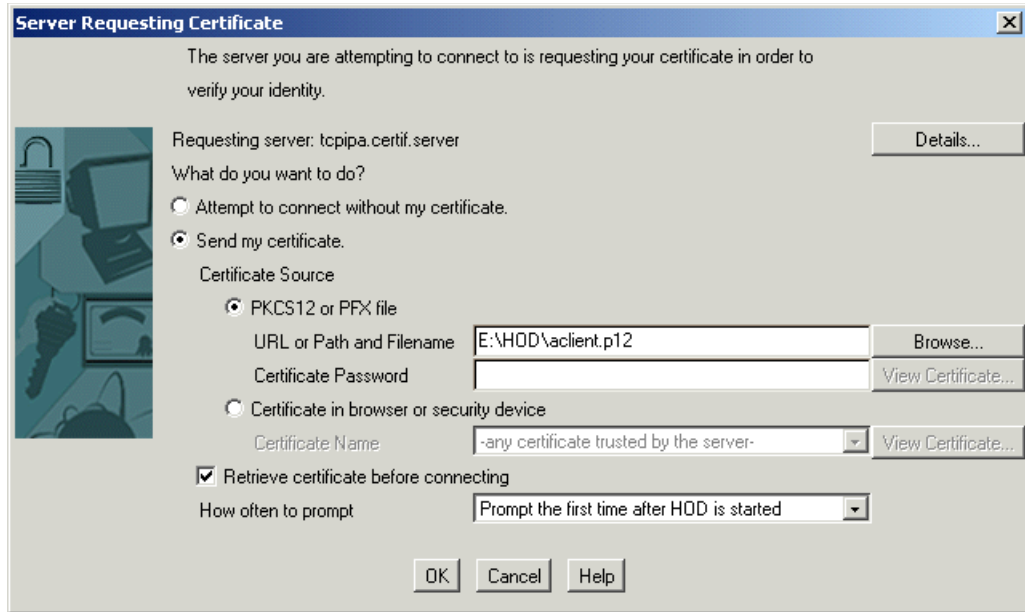


Figure 12-23 HOD PKCS12 password prompt

6. The new session will start and you will receive the application or USSTAB customized in TCP/IP profile data set (in our case, TSO).

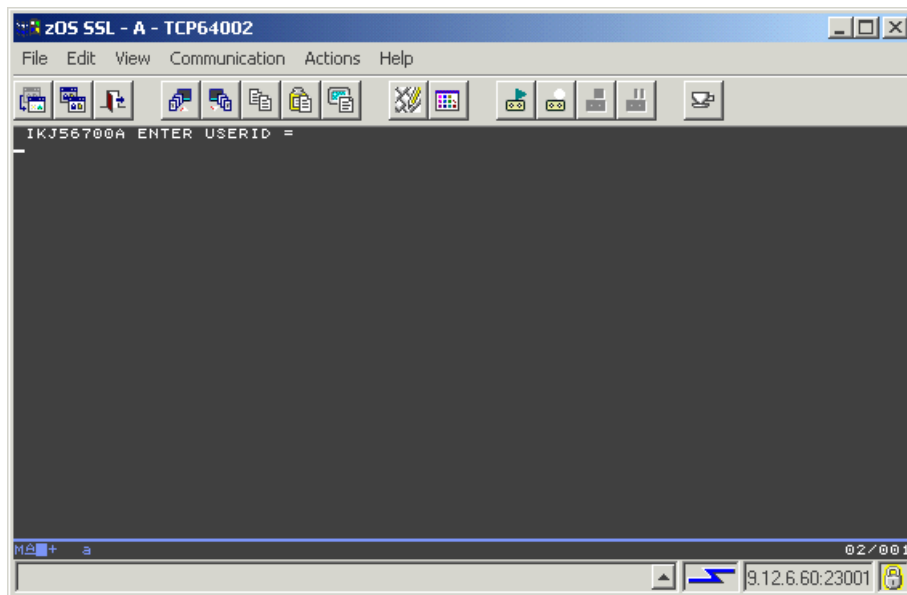


Figure 12-24 HOD SSL session started

Now you can check the session characteristics using the commands in Figure 12-25 on page 322.

```

D TCPIP,TCPIPA,T,CONN
EZZ6064I TELNET CONNECTION DISPLAY 369
      EN                      TSP
CONN  TY IPADDR..PORT      LUNAME  APPLID  PTR LOGMODE
-----
0000024C 4S 9.24.106.91..1225 1 TCP64002 SC64TS07 TAE D4C32XX3
-----
----- PORT: 23001 ACTIVE          PROF: CURR CONNS: 1
-----
3 OF 3 RECORDS DISPLAYED

```

Figure 12-25 Display CONN

1 There is a connection opened with PORT 23001 and the connection number is 24C.

```

D TCPIP,TCPIPA,T,CONN,CO=24C
EZZ6065I TELNET CONNECTION DISPLAY 371
CONN: 0000024C 2 CLNTIP..PORT: 9.24.106.91..1225
LINKNAME: OSA22EOLINK DESTIP..PORT: 9.12.6.60..23001
HOSTNAME: NO HOSTNAME
CONNECTED: 13:41:35 09/27/2001 STATUS: SESSION ACTIVE
PORT: 23001 ACTIVE SECURE ACCESS: SECURE 4S SAFCERT 3
PROTOCOL: TN3270E LOGMODE: D4C32XX3 DEVICETYPE: IBM-3278-3-E
OPTIONS: ETET--- 3270E FUNCTIONS: BSR----
NEWENV FUNCTIONS: --
USERIDS
  RESTRICTAPPL: **N/A** CLIENTAUTH: FASCINI 4
  EXPRESSLOGON: **N/A**
APPL: SC64TS07
LUNAME: TCP64002 TYPE: TERMINAL GENERIC
MAPS CONN IDENTIFIER OBJECT DEFAPPL OPTIONS
LU MAPPINGS:
  >*DEFLUS* **N/A** -----
DEFAULTAPPL:
  NL (NULL) TSO -----
USS TABLE: **N/A**
INT TABLE: **N/A**
PARMS:
PERS FUNCT DIA SECURE TIMERS SMF MAX LINE
(LMTQ) (OATSSWH) (DRF) (SCKLECX) (IKPSTS) (ITIT) (RSQ) (BDCTT)
----
---- --TS--- --- -B--D-- ---STS ---- RSQ --C-- *DEFAULT
---- ----- --- --S---- ----- ---- --- ----- *TGLOBAL
---- ----- --- --S---- ----- ---- --- ----- *TPARMS
---- --TS--- DJ- SSS-DF- ---STS ---- RSQ --C-- TP-CURR
---- --TS--- DJ- SSS-DF- ---STS ---- RSQ --C-- FINAL
29 OF 29 RECORDS DISPLAYED

```

Figure 12-26 Display CONN,CO=24C

- 2 The same connection is displayed with more details.
- 3 This connection is using SSL and the certificate was checked by RACF.
- 4 The RACF user ID associated with this connection is the same user ID used in RACDCERT commands.

TN3270 client (HOD) negotiated-Telnet configuration

To configure a negotiated-Telnet session using IBM's Host On-Demand client (HOD), click Yes on **Telnet-Negotiated** (in the Security tab), as illustrated in Figure 12-27, and then click **OK**.

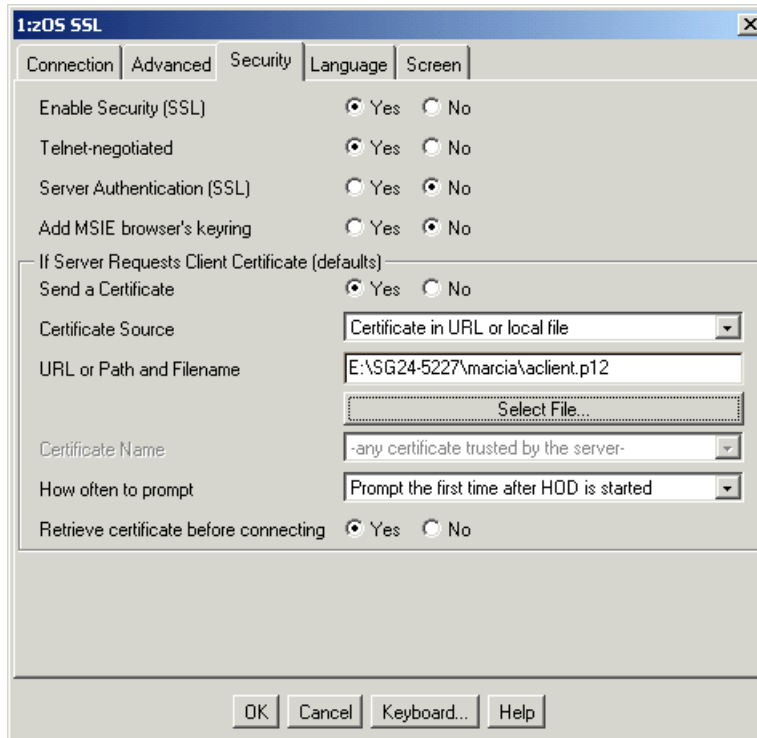


Figure 12-27 HOD TLS definition

When you start the HOD session, the SSL handshake will be negotiated. You can see the difference between a negotiated connection and a non-negotiated connection in the figures that follow. In all of the tests, SSLTIMEOUT 5 was used:

► CONNTYPE NEGOTSECURE

A TN3270 negotiation in which the client determines if it is willing to enter into a secure connection. If the client agrees, SSL protocols are used for all subsequent communication. If the client does not agree, the connection is closed.

- HOD Telnet-Negotiated = YES displays Figure 12-28.

```
EZZ6034I TELNET CONN 00000BB LU **N/A** ACCEPTED 23001 624
IPADDR..PORT 9.24.106.91..1260
EZZ6034I TELNET CONN 00000BB LU TCP64002 NEGOTIATED TN3270E 625
IPADDR..PORT 9.24.106.91..1260
LOGON
EZZ6034I TELNET CONN 00000BB LU TCP64002 IN SESSION SC64TS03 627
IPADDR..PORT 9.24.106.91..1260
```

Figure 12-28 NEGOTSECURE and Telnet-Negotiated

- HOD Telnet-Negotiated = NO. The session is not established (see Figure 12-29 on page 324).

```

EZZ6034I TELNET CONN 000000C0 LU **N/A** ACCEPTED 23001 640
          IPADDR..PORT 9.24.106.91..1264
EZZ6035I TELNET DEBUG CLIENT IPADDR..PORT 9.24.106.91..1264 641
CONN: 000000C0 LU:          MOD: EZBTTRCV
RCODE: 100B-00 Unexpected SSL handshake encountered.
PARM1: 00000000 PARM2: 00000000 PARM3: 00000000
EZZ6034I TELNET CONN 000000C0 LU **N/A** CONN DROP ERR 100B 642
          IPADDR..PORT 9.24.106.91..1264          EZBTTRCV

```

Figure 12-29 *NEGTSECURE and no Telnet-Negotiated*

► CONNTYPE SECURE

The traditional SSL handshake is used to start the SSL connection. If the client does not start the handshake within the time specified by SSLTIMEOUT, an attempt is made to do a negotiated SSL handshake. If the client rejects the negotiated attempt, the connection is closed.

- HOD Telnet-Negotiated = YES (see Figure 12-30).

```

EZZ6034I TELNET CONN 00000032 LU **N/A** ACCEPTED 23001 746
          IPADDR..PORT 9.24.106.91..1290
EZZ6035I TELNET DEBUG CLIENT IPADDR..PORT 9.24.106.91..1290 747
CONN: 00000032 LU:          MOD: EZBTTSMT
RCODE: 600E-00 SSL handshake Timed out.
PARM1: FFFFFFFEA PARM2: 00000000 PARM3: 00000000
EZZ6034I TELNET CONN 00000032 LU TCP64002 NEGOTIATED TN3270E 748
          IPADDR..PORT 9.24.106.91..1290
LOGON
EZZ6034I TELNET CONN 00000032 LU TCP64002 IN SESSION SC64TS03 750
          IPADDR..PORT 9.24.106.91..1290

```

Figure 12-30 *Secure and Telnet-Negotiated*

- HOD Telnet-Negotiated = NO (see Figure 12-31).

```

EZZ6034I TELNET CONN 00000038 LU **N/A** ACCEPTED 23001 757
          IPADDR..PORT 9.24.106.91..1293
EZZ6034I TELNET CONN 00000038 LU TCP64002 NEGOTIATED TN3270E 758
          IPADDR..PORT 9.24.106.91..1293
LOGON
EZZ6034I TELNET CONN 00000038 LU TCP64002 IN SESSION SC64TS03 760
          IPADDR..PORT 9.24.106.91..1293

```

Figure 12-31 *Secure and no Telnet-Negotiated*

► CONNTYPE ANY

The client can connect as secure or basic. Telnet first tries a standard SSL handshake. If the handshake times out, negotiated SSL attempted.

- HOD Telnet-Negotiated = YES (Figure 12-32 on page 325).

```

EZZ6034I TELNET CONN 00000032 LU **N/A** ACCEPTED 23001 005
          IPADDR..PORT 9.24.106.91..1331
EZZ6035I TELNET DEBUG CLIENT IPADDR..PORT 9.24.106.91..1331 006
CONN: 00000032 LU:          MOD: EZBTTSMT
RCODE: 600E-00 SSL handshake Timed out.
PARM1: FFFFFFFEA PARM2: 00000000 PARM3: 00000000
EZZ6034I TELNET CONN 00000032 LU TCP64002 NEGOTIATED TN3270E 007
          IPADDR..PORT 9.24.106.91..1331
LOGON
EZZ6034I TELNET CONN 00000032 LU TCP64002 IN SESSION SC64TS03 009
          IPADDR..PORT 9.24.106.91..1331

```

Figure 12-32 Any and Telnet-Negotiated

- HOD Telnet-Negotiated = NO.

```

EZZ6034I TELNET CONN 0000002B LU **N/A** ACCEPTED 23001 995
          IPADDR..PORT 9.24.106.91..1328
EZZ6034I TELNET CONN 0000002B LU TCP64002 NEGOTIATED TN3270E 996
          IPADDR..PORT 9.24.106.91..1328
LOGON
EZZ6034I TELNET CONN 0000002B LU TCP64002 IN SESSION SC64TS03 998
          IPADDR..PORT 9.24.106.91..1328

```

Figure 12-33 Any and no Telnet-Negotiated

► CONNTYPE BASIC

A basic (non-SSL) connection is used.

- Enable Security = YES.

```

EZZ6034I TELNET CONN 00000044 LU **N/A** ACCEPTED 23001 171
          IPADDR..PORT 9.24.106.91..1357
EZZ6035I TELNET DEBUG CLIENT IPADDR..PORT 9.24.106.91..1357 172
CONN: 00000044 LU:          MOD: EZBTTRCV
RCODE: 100B-00 Unexpected SSL handshake encountered.
PARM1: 00000000 PARM2: 00000000 PARM3: 00000000
EZZ6034I TELNET CONN 00000044 LU **N/A** CONN DROP ERR 100B 173
          IPADDR..PORT 9.24.106.91..1357          EZBTTRCV

```

Figure 12-34 Basic and SSL

- HOD Enable Security = NO.

```

EZZ6034I TELNET CONN 0000003D LU **N/A** ACCEPTED 23001 149
          IPADDR..PORT 9.24.106.91..1351
EZZ6034I TELNET CONN 0000003D LU TCP64002 NEGOTIATED TN3270E 150
          IPADDR..PORT 9.24.106.91..1351
LOGON
EZZ6034I TELNET CONN 0000003D LU TCP64002 IN SESSION SC64TS07 152
          IPADDR..PORT 9.24.106.91..1351

```

Figure 12-35 Basic and no SSL

- ▶ CONNTYPE NONE
 - Any client connection request is rejected.
 - HOD Enable Security = YES.

```
EZZ6034I TELNET CONN 0000021 LU **N/A** ACCEPTED 23001 268
          IPADDR..PORT 9.24.106.91..1365
EZZ6035I TELNET DEBUG CLIENT IPADDR..PORT 9.24.106.91..1365 269
CONN: 0000021 LU:          MOD: EZBTACP
RCODE: 100A-00 Connection type of NONE was specified.
PARM1: 00000000 PARM2: 00000000 PARM3: 00000000
EZZ6034I TELNET CONN 0000021 LU **N/A** CONN DROP ERR 100A 270
          IPADDR..PORT 9.24.106.91..1365          EZBTACP
```

Figure 12-36 None and SSL

- HOD Enable Security = NO.

```
EZZ6034I TELNET CONN 0000028 LU **N/A** ACCEPTED 23001 282
          IPADDR..PORT 9.24.106.91..1372
EZZ6035I TELNET DEBUG CLIENT IPADDR..PORT 9.24.106.91..1372 283
CONN: 0000028 LU:          MOD: EZBTACP
RCODE: 100A-00 Connection type of NONE was specified.
PARM1: 00000000 PARM2: 00000000 PARM3: 00000000
EZZ6034I TELNET CONN 0000028 LU **N/A** CONN DROP ERR 100A 284
          IPADDR..PORT 9.24.106.91..1372          EZBTACP
```

Figure 12-37 None and no SSL

12.3 Express Logon Feature (ELF)

Express Logon Feature (ELF) was introduced in IBM Communications Server for OS/390 V2R10 IP Services. ELF allows a user on a workstation, with a TN3270 client and an X.509 certificate, to log on to an SNA application without entering a user ID or password.

Express Logon Feature allows users to:

- ▶ Reduce the time administrators spend maintaining user IDs and passwords.
- ▶ Reduce the number of user IDs and passwords that users have to remember.
- ▶ Remove a potential security risk of users writing down user IDs and passwords, or sharing them with someone else.

The Express Logon Feature is supported on two-tier and three-tier network designs. The two-tier design utilizes the z/OS TN3270 Telnet server. The three-tier design utilizes a middle-tier TN3270 server and a Digital Certificate Access Server (DCAS) running on the z/OS.

In order for an application to be accessed using the Express Logon Feature, a PassTicket data class profile (PTKTDATA) must be defined on each target RACF system (that is, the host where DCAS is running, and any host where RACF and a target application is located).

Both network designs require a TN3270 client workstation that supports Secure Sockets Layer (SSL) connections with client authentication and an X.509 certificate. Using RACF services in z/OS, the client certificate must be associated with a valid user ID. The only client-side product that supports the Express Logon Feature is the IBM WebSphere Host On-Demand V5.0 or V6.0.

12.3.1 Two-tier network design

In the two-tier design, the user starts an SSL connection with level 2 client authentication, which passes the client certificate to the MVS host TN3270 server. Level 2 client authentication is required because the client certificate has to be associated in RACF with a valid user ID of the SNA application that the user wants to log onto. The MVS host TN3270 server uses RACF Secure Sign-on services to obtain a user ID and PassTicket.

The two-tier design is supported in z/OS V1R2 and OS/390 V2R10 + PQ47742 (UQ55691).

Figure 12-38 shows the data flow using ELF two-tier network design.

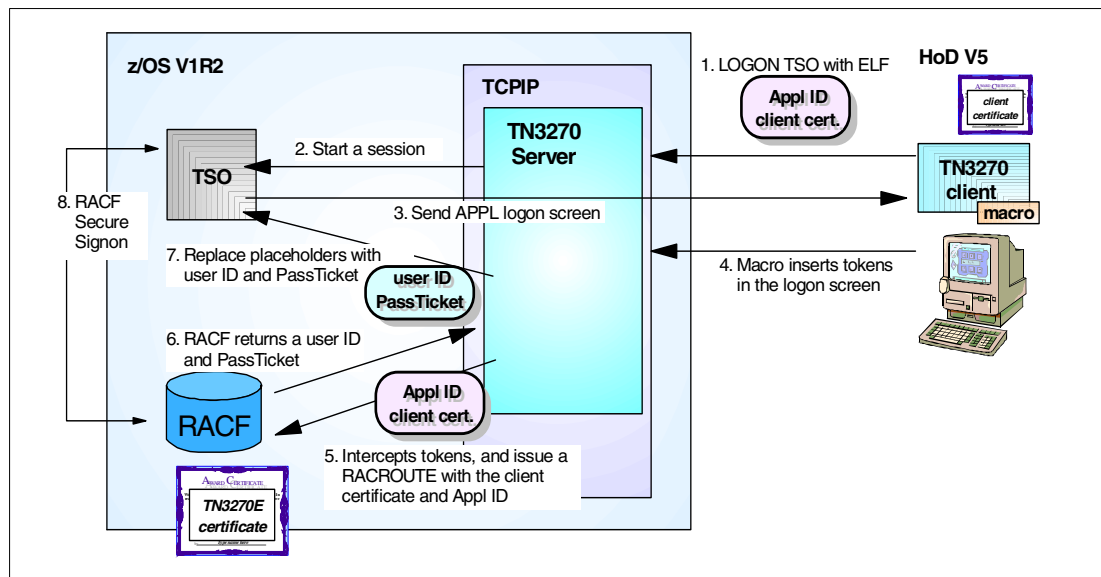


Figure 12-38 ELF two-tier network design

The figure shows the steps where a client accesses TSO session on z/OS:

1. The user has an HOD icon that starts an emulator session configured to use SSL client authentication. The session has a macro associated with it. A client certificate has to be available from the terminal and presented to the TN3270 server for the SSL handshake. During the SSL handshake, the client certificate is passed to the TN3270 server and validated. During Telnet function negotiation, the ELF capability is negotiated using RFC 1572.
2. The application ID is sent from the client to the TN3270 server and the server starts the session with TSO. The application ID must match the profile-name defined for PassTicket in RACF. See step 3 in 12.3.3, “Implementing ELF in a two-tier design” on page 330.
3. The logon screens come to the emulator.
4. The macro runs and inserts placeholder strings in the user ID and password fields.
5. The TN3270 server intercepts the placeholder strings and sends the certificate and the target application ID to RACF.

6. RACF converts the HOD client's certificate to a TSO user ID and PassTicket and sends them back to the TN3270 server.
7. The TN3270 server inserts the user ID and PassTicket into the 3270 datastream at the macro-inserted placeholder locations and sends it to the application.
8. The application presents the user ID and PassTicket to RACF (or other compatible host access control facility), which approves them, and the logon completes as usual.

For configuration of a two-tier design see 12.3.3, "Implementing ELF in a two-tier design" on page 330.

12.3.2 Three-tier network design

In the three-tier design, the user starts the TN3270 connection to the middle-tier server. This was the original design of ELF because Telnet Server on OS/390 did not support Digital Certificate Access Requester (DCAR). For the client, the ELF customizing is the same for two-tier and three-tier designs.

The three-tier components are:

- ▶ A client workstation that supports Secure Sockets Layer (SSL) connections with client authentication and an X.509 certificate.
- ▶ A middle-tier TN3270 server, so called because it does not reside on the host, but rather between the client and the host. This server communicates with a DCAS using an SSL connection with client authentication. It sends the user's certificate from the workstation and an application ID to the DCAS and expects to receive a user ID and PassTicket (a one-time password) in response. This is the user ID and password that will be used to log on to the SNA application.
- ▶ The Digital Certificate Access Server (DCAS) resides on the host. The DCAS uses RACF services to obtain a user ID that is associated with the certificate sent by the client. RACF also provides Secure Sign-on services, which the DCAS uses to generate a PassTicket. A PassTicket is a RACF token similar to a password except that it is valid only for 10 minutes.

There must be a DCAR (Digital Certificate Access Requester) in the Telnet Server and a DCAS (Digital Certificate Access Server) in the host.

The term DCAR is used to describe the part of the TN3270 middle-tier server that supports the Express Logon Feature and communicates as a client with the DCAS. It is not separate from the TN3270 middle-tier server.

The DCAS's client is the middle-tier TN3270 server or DCAR, which attempts to log on to an SNA application for the workstation client. The DCAS receives a digital certificate from the DCAR and returns a user ID and PassTicket. SSL communication is used between the DCAS and the DCAR. The server recognizes that the client wants the Express Logon function and invokes the DCAR, which opens an SSL connection with client authentication and passes the workstation's certificate and application name to the DCAS on the host. The DCAS uses RACF Secure Sign-on services to obtain a user ID and PassTicket, which the DCAS returns to the DCAR. The DCAR passes this information back to the TN3270 server. A PassTicket is a RACF token similar to a password except that it is valid only for 10 minutes.

The middle-tier IBM TN3270 servers supporting Express Logon are:

- ▶ CS2 6.1
- ▶ CS/NT 6.1.1 PTF
- ▶ CS/AIX 6.0.0.1 PTF

The SNA connection between the TN3270 server and SNA application can be SNA LU2, DLUR, HPR/IP (EE), or AnyNet connection.

Figure 12-39 shows the data flow using ELF three-tier network design.

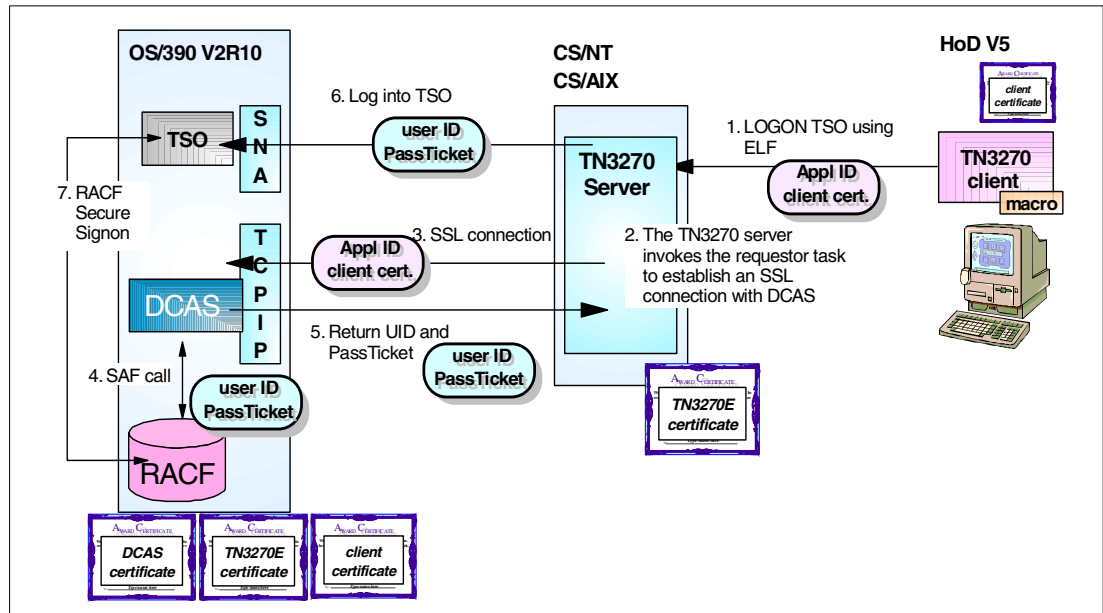


Figure 12-39 ELF three-tier network design

The following are the steps executed when a client accesses a TSO session on z/OS:

1. The user has an HOD icon that starts an emulator session configured to use SSL client authentication. The session has a macro associated with it. A client certificate has to be available from the terminal and presented to the TN3270 server for the SSL handshake. During the SSL handshake, the client certificate is passed to the TN3270 server and validated. During Telnet function negotiation, the ELF capability is negotiated using RFC 1572.

The application ID is sent from the client to the TN3270 server. The logon screens come to the emulator as usual, but the macro plays and inserts placeholder strings in the user ID and password fields. The TN3270 server intercepts the placeholder strings.

2. Once the application ID and client certificate are received by the TN3270 server, it invokes the DCAR function to establish the SSL communication to the DCAS server. The TN3270 server's certificate has to be sent to DCAS and authenticated.
3. The TN3270 server sends the certificate and the target application ID to DCAS on the z/OS host over a secure, trusted connection.
4. The DCAS server makes SAF calls to convert the HOD client's certificate to a TSO user ID and PassTicket.
5. The DCAS server sends the user ID and PassTicket back to the TN3270 server.
6. The TN3270 server inserts the user ID and PassTicket into the 3270 datastream at the macro-inserted placeholder locations and sends it to the application.
7. The application presents the user ID and PassTicket to RACF (or other compatible host access control facility), which approves them and the logon completes as usual.

For configuration steps of a three-tier design see 12.3.4, "Implementing ELF in a three-tier design" on page 336.

The ELF two-tier design implementation is simpler than the three-tier design implementation, because you no longer need a DCAS for a middle-tier TN3270 server. Choose the three-tier design if you do not want to have a TN3270 server on z/OS or if you are going to use Host Publisher. Host Publisher acts as the client and the middle-tier server together.

12.3.3 Implementing ELF in a two-tier design

Follow the steps below to implement ELF with the two-tier design, using HOD as the client and accessing TSO on z/OS:

1. Define an SSL session with client authentication level 2 (CLIENTAUTH SAFCERT in TCP/IP profile). The procedure to define an SSL session is in 12.1.1, “TN3270 configuration parameters for SSL” on page 301.
2. Define the EXPRESSLOGON parameter on TCP/IP profile. The ITSO profile statements used for ELF are shown in Figure 12-40.

```
TELNETPARMS
SECUREPORT 23003
KEYRING SAF tcpipa.tn3270.keyring
CONNTYPE SECURE
CLIENTAUTH SAFCERT
EXPRESSLOGON
DEBUG DETAIL
ENDTELNETPARMS
```

Figure 12-40 Profile definition for ELF

3. Define the PassTicket profile to RACF. For each application to which users are to gain access with a PassTicket, you must define a PTKTDATA class profile. In our example, the application is TSO and the commands issued are shown in Figure 12-41.

```
SETROPTS CLASSACT(PTKTDATA)
SETROPTS RACLIST(PTKTDATA)
RDEFINE PTKTDATA TSOSC64 SSIGNON(KEYMASKED(E6C9D30195D4C1E7)) UACC(NONE)
SETR RACLIST(PTKTDATA) REFRESH
```

Figure 12-41 RACF definition for PassTicket

The profile name (TSOSC64 in our case) must match the application ID configured on the HOD client window. For TSO, the rule to create a profile name is: TSO+smfid.

Define a key using KEYMASKED, even though the value is not significant. *This is required!*

For more details about PTKTDATA and rules of profile names, see *z/OS V1R2.0 SecureWay Security Server RACF Security Administrator's Guide*.

4. Now, after doing the previous steps, start TCP/IP and establish a session with port 23003 to create the HOD macro for ELF. In this case we have defined DEFAULTAPPL TSO and received the TSO User ID window.
 - a. Click **Actions** -> **Record Macro** or click the **Record Macro** icon.

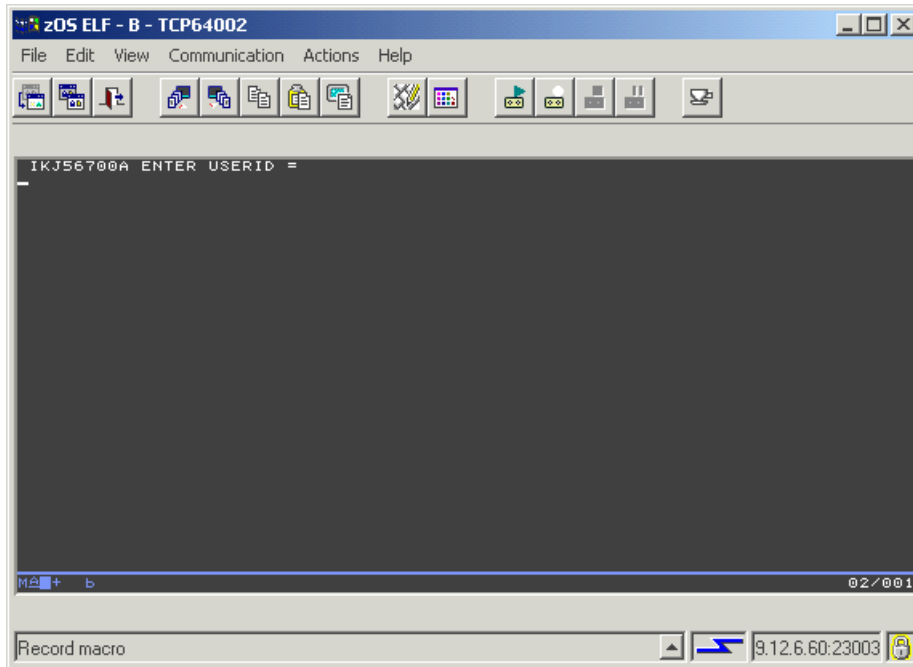


Figure 12-42 Start recording HOD macro

- b. Chose **New**, specify the name and the description of the macro and click the **Express Logon Feature** box. Then, click **OK**, as shown in Figure 12-43.

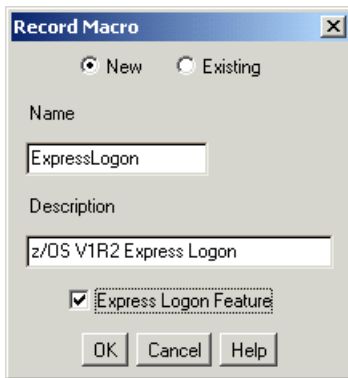


Figure 12-43 HOD macro - Record Macro

- c. Specify the Application ID (the same used in the RACF command) and click **OK**.

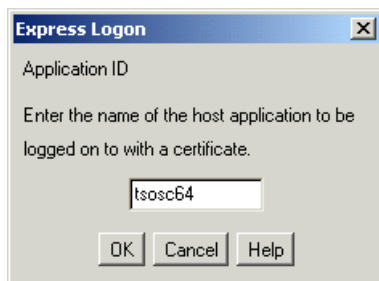


Figure 12-44 HOD macro - Application ID

- d. Since the TSO screen contains a user ID field, click **OK** on this window, as Figure 12-45 shows.



Figure 12-45 HOD macro -Selection Criteria

- e. In the window illustrated in Figure 12-46, click **No** and **Next>**.

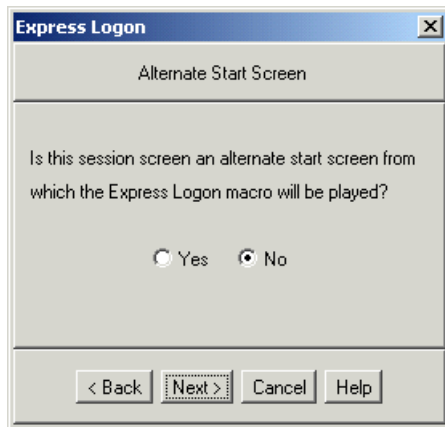


Figure 12-46 HOD macro - Alternate Start Screen

- f. In the window in Figure 12-47, click **Yes** and **Next>**.

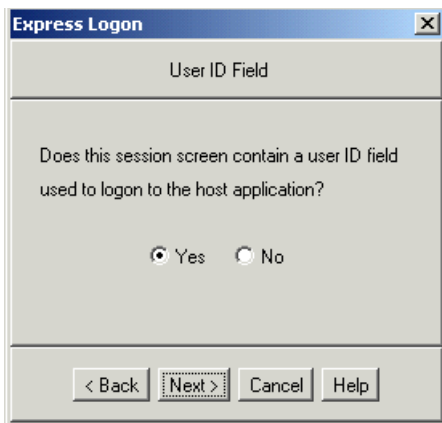


Figure 12-47 HOD macro - User ID Field

- g. Click **Current** and the Row and Column fields will be filled in. Then, specify your RACF user ID in the User ID field. Click **Next>**, as shown in Figure 12-48.

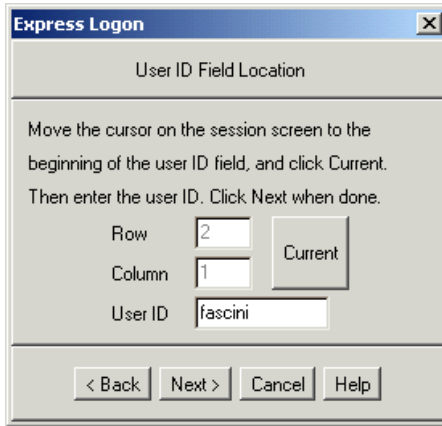


Figure 12-48 HOD macro - User ID Field Location

- h. Since the session screen does not contain a password field, click **No** and **Next>**, as illustrated in Figure 12-49.



Figure 12-49 HOD macro - Password Field

- i. Go to TSO screen, press Enter, and click **OK** in the Express Logon window:

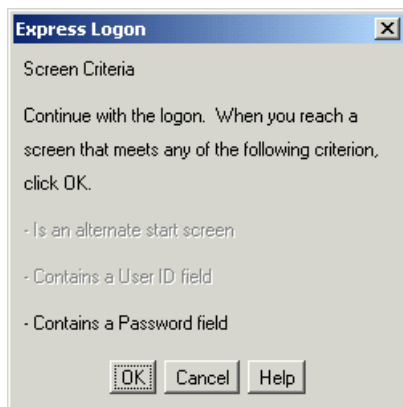


Figure 12-50 HOD macro - Screen Criteria

- j. Click **Current** and the Row and Column fields will be filled in. Then, specify your RACF password in the Password field. Click **Finish**.

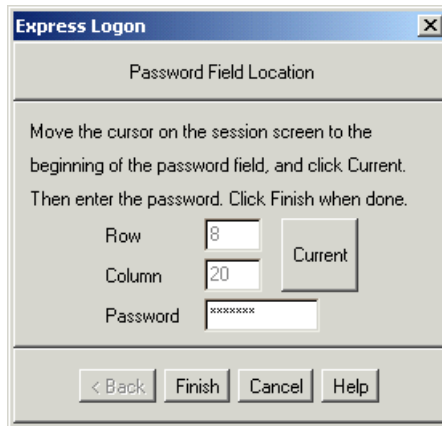


Figure 12-51 HOD macro - Password Field Location

- k. Click **OK** on next window, press Enter on the TSO screen and click **Stop Macro** icon (or click **Actions -> Stop Macro**).
- l. Now, click **Play Macro** icon (or **Actions -> Play Macro**) and chose the macro's name you have just created, as illustrated in Figure 12-52.

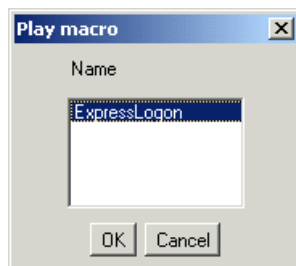


Figure 12-52 HOD macro's name

- m. Click **OK**. The macro will run and you will receive TSO's main menu, as shown in Figure 12-53 on page 335.

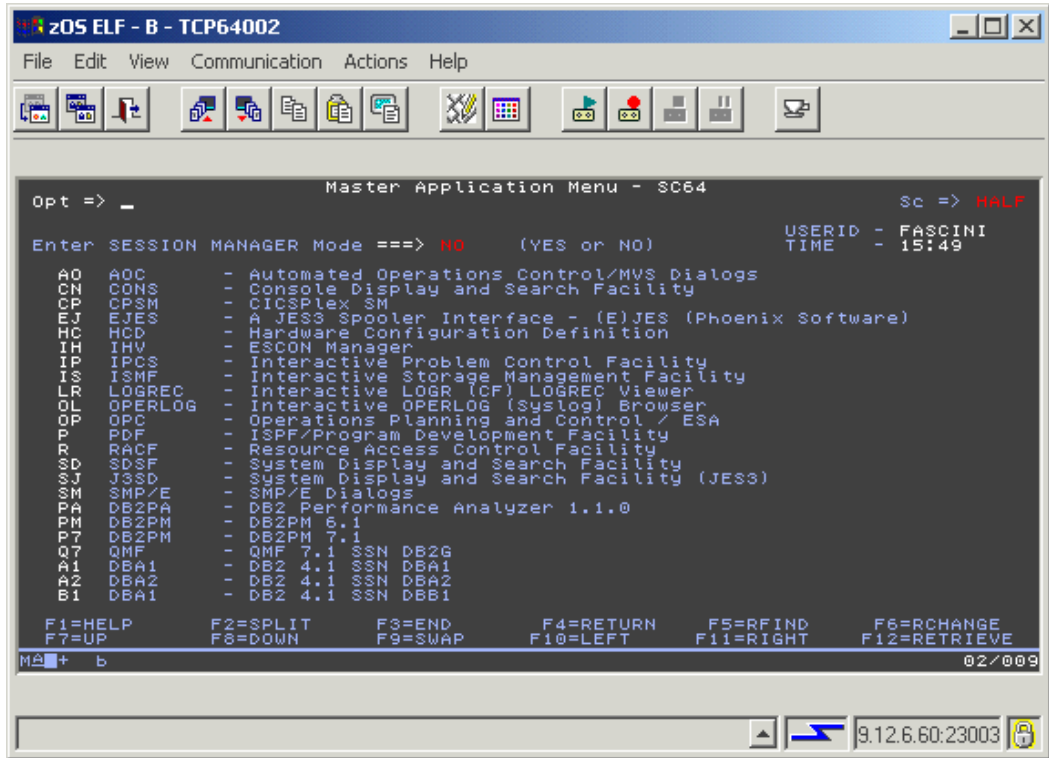


Figure 12-53 TSO's main menu

You can display the connection to check that ELF is being used.

```

D TCPIP,TCPIPA,T,CONN
EZZ6064I TELNET CONNECTION DISPLAY 664
      EN                                TSP
CONN  TY IPADDR..PORT                LUNAME  APPLID  PTR LOGMODE
-----
00000F76 4S 9.24.106.91..1347        TCP64002 SC64TS05 TAE D4C32XX3
-----
----- PORT: 23003 ACTIVE                PROF: CURR CONNS: 1
-----
3 OF 3 RECORDS DISPLAYED

```

Figure 12-54 Connection display

```

D TCPIP,TCPIPA,T,CONN,CO=F76
EZZ6065I TELNET CONNECTION DISPLAY 689
CONN: 0000F76          CLNTIP..PORT: 9.24.106.91..1347
LINKNAME: OSA22EOLINK  DESTIP..PORT: 9.12.6.60..23003
HOSTNAME: NO HOSTNAME
CONNECTED: 15:43:14 09/28/2001 STATUS: SESSION ACTIVE
PORT: 23003 ACTIVE SECURE ACCESS: SECURE 4S SAFCERT 1
PROTOCOL: TN3270E LOGMODE: D4C32XX3 DEVICETYPE: IBM-3278-3-E
  OPTIONS: ETET--- 3270E FUNCTIONS: BSR----
           NEWENV FUNCTIONS: E-

USERIDS
  RESTRICTAPPL: **N/A**  CLIENTAUTH: FASCINI 2
  EXPRESSLOGON: FASCINI 3
APPL: SC64TS05
LUNAME: TCP64002 TYPE: TERMINAL GENERIC
MAPS CONN IDENTIFIER OBJECT DEFAPPL OPTIONS
LU MAPPINGS:
                                     >*DEFLUS* **N/A**
                                     -----
  DEFAULTAPPL:
    NL (NULL) TSO
                                     -----
  USS TABLE: **N/A**
  INT TABLE: **N/A**
  PARMS:
PERS FUNCT DIA SECURE TIMERS SMF MAX LINE
(LMTQ) (OATSSWH) (DRF) (SCKLECX) (IKPSTS) (ITIT) (RSQ) (BDCTT)
----
---- --TS--- --- -B--D-- ---STS --- RSQ --C-- *DEFAULT
----
----
---- --S-----
----
----
---- --S-----
----
----
---- --TS--- DJ- SSS-DFX ---STS --- RSQ --C-- TP-CURR
---- --TS--- DJ- SSS-DFX ---STS --- RSQ --C-- FINAL
29 OF 29 RECORDS DISPLAYED

```

Figure 12-55 Connection display - detail

1, **2** CLIENTAUTH level 2 is being used for Express Logon.

3 This is the RACF user ID associated with the client certificate.

12.3.4 Implementing ELF in a three-tier design

The implementation of ELF with three-tier design is the same in both z/OS V1R2 and OS/390 V2R10. The following sections describe the implementation.

DCAS - DCAR connection

The Digital Certificate Access Server (DCAS) is a TCP/IP server that runs on OS/390 V2R10 and later. The middle-tier TN3270 servers connect to DCAS using Secure Sockets Layer V3 (SSL). The purpose of DCAS is to receive an application ID and a digital certificate from a middle-tier TN3270 server, then ask RACF to return a valid user ID that has been associated with the certificate and to generate a PassTicket for the input user ID and application ID.

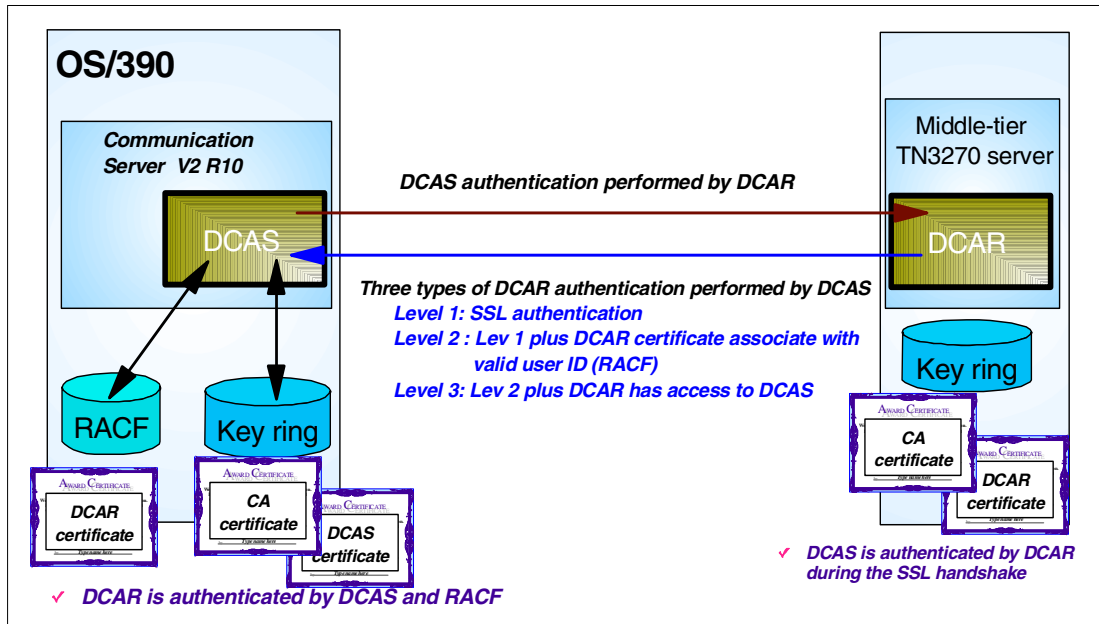


Figure 12-56 DCAS / DCAR

Authenticating the DCAS

The DCAS authentication is always performed by the DCAR during the SSL handshake. Authentication requires that the DCAS have a private key and an associated X.509 digital certificate defined in a keyring.

If you use a self-signed certificate, it has to be treated as a CA certificate by all TN3270 servers. Follow the steps below:

1. Export the DCAS self-signed certificate into a file in the DER binary format.
2. Send it to a TN3270 server, using FTP with the BINARY send option.
3. Store the certificate into a key database for the TN3270 server as a trusted Certificate Authority.

For more information about certificates, see 2.2.4, “Digital certificates” on page 13 and Appendix 10, “Certificate management in z/OS” on page 203.

Authenticating the DCAR

The DCAR is the client that interacts with the DCAS. Authenticating the DCAR involves additional levels of control in which the client must have a key database with a certificate. Depending on the control level, the certificate is authenticated by SSL and the DCAS using RACF services.

There are three levels of client authentication from which to choose:

► Level 1

With Level 1 authentication, the DCAS uses the client authentication provided by SSL at the time of the SSL handshake. The keyring used by the DCAS must contain the following certificates:

- The DCAS certificate.
- The certificate of a CA that has signed the TN3270 server certificate, or the TN3270 certificate itself, if a self-signed certificate is used for the TN3270 server.

To configure DCAS for this level of authentication, specify the CLIENTAUTH LOCAL1 keyword in the DCAS configuration file. Use the KEYRING or the SAFKEYRING keywords in the DCAS configuration file to specify the keyring used by the DCAS.

► Level 2

Level 2 includes Level 1 authentication plus additional verification that the DCAR certificate has been associated in RACF with a valid user ID. To configure DCAS for this level of authentication, specify the CLIENTAUTH LOCAL2 keyword in the DCAS configuration file. Use FTP (with the BINARY send option) to send the client's DER certificate to an MVS data set. Use the RACDCERT ADD command to add the certificate to RACF and associate it with a user ID, as shown in Figure 12-57.

```
RACDCERT ID(dcasid) ASID('DCAS.DCAR.CERT') TRUST
```

Figure 12-57 DCAS certificate trusted

► Level 3

Level 3 includes level 2 authentication plus it verifies that the DCAR has access to the DCAS. The user ID derived from the certificate using the RACF checks from Level 2 is defined as having access to the SERVAUTH RACF class and the EZA.DCAS.cvtsysname resource in the SERVAUTH class. The following conditions apply:

- If the SERVAUTH class is not active or the EZA.DCAS.cvtsysname profile is not defined, or both, it is assumed this enhanced level is not requested.
- If the SERVAUTH class is active and the EZA.DCAS.cvtsysname profile is defined (but not for the user associated with the certificate) the requestor's connection is terminated.

Use the commands in Figure 12-58 to create the RACF profile and give the access permission to a user.

```
RDEFINE SERVAUTH EZA.DCAS.cvtsysname UACC(NONE)  
PERMIT EZA.DCAS.cvtsysname CLASS(SERVAUTH) ACCESS (CONTROL) ID(dcasid)  
SETR RACLIST(SERVAUTH) REFRESH
```

Figure 12-58 RACF commands for DCAS

To configure DCAS for Level 3 authentication, follow these steps:

- Specify the CLIENTAUTH LOCAL2 keyword and value in the DCAS configuration file.
- Activate the SERVAUTH RACF class.
- Define a profile for the EZA.DCAS.cvtsysname resource and associate the profile with the user ID associated with the certificate.

Note: The ID associated with the certificate and the EZA.DCAS.cvtsysname can be any valid user ID.

DCAS customization

Follow these steps to customize DCAS for ELF with a three-tier design:

1. Define an SSL session between DCAS (z/OS) and DCAR (middle-tier Telnet Server) and between DCAR and TN3270 client (HOD in our case). The procedure to define an SSL

session is explained in 12.1.1, “TN3270 configuration parameters for SSL” on page 301. Instead of using the TCP/IP name, use the DCAS name on RACF commands.

The user ID associated with the keyring and the DCAS server’s certificate has to be user defined in the STARTED procedure of DCAS.

The DCAS certificate has to be imported into the key database used by the TN3270 server (middle-tier) and defined as trusted.

The HOD client certificate has to be defined in RACF.

2. Set up DCAS to use RACF services.

- Define the STARTED profile and OPERCMDS.

```
ADDUSER DCAS DFLTGRP(OMVSGRP) OMVS(UID(0) HOME('/'))

RDEFINE STARTED DCAS.* STDATA(USER(DCAS))
SETROPTS RACLIST (STARTED) REFRESH

RDEFINE OPERCMDS(MVS.SERVGR.DCAS) UACC(NONE)
PERMIT MVS.SERVGR.DCAS CLASS(OPERCMDS) ACCESS(CONTROL) ID(DCAS)
SETROPTS RACLIST(OPERCMDS) REFRESH
```

Figure 12-59 DCAS started class and OPERCMDS

- Permit the DCAS to use certificate services.

```
SETROPTS CLASSACT(DIGTCERT DIGTRING)
RDEFINE FACILITY(IRR.DIGTCERT.LIST) UACC(NONE)
RDEFINE FACILITY(IRR.DIGTCERT.LISTRING) UACC(NONE)
PERMIT IRR.DIGTCERT.LIST CLASS(FACILITY) ID(DCAS) ACCESS(CONTROL)
PERMIT IRR.DIGTCERT.LISTRING CLASS(FACILITY) ID(DCAS) ACCESS(CONTROL)
SETROPTS RACLIST(DIGTRING DIGTCERT) REFRESH
```

Figure 12-60 Certificate commands

- Define PassTicket data profile.

```
SETROPTS CLASSACT(PTKTDATA)
SETROPTS RACLIST(PTKTDATA)
RDEFINE PTKTDATA TSORA03 SSIGNON(KEYMASKED(E6C9D30195D4C1E7)) UACC(NONE)
SETROPTS RACLIST(PTKTDATA) REFRESH
```

Figure 12-61 PassTicket commands

3. Define DCAS configuration file.

Some of the configuration parameters you can use in the DCAS configuration file are shown in Table 12-1.

Table 12-1 DCAS configuration parameters

Parameters	Description
IPADDR	Allows you to define the IP address to which the DCAS will bind.
PORT	Defines the port number on which DCAS will run.
KEYRING ¶	Defines the HFS key database file containing the certificate to be used during the SSL handshake.

Parameters	Description
STASHFILE	Specifies the password file to the associate key database file.
SAFKEYRING ¶	Defines the RACF-defined keyring containing the certificate to be used during the SSL handshake.
V3CIPHER	Specifies a subset of the supported SSL V3 cipher algorithms.

¶ The keywords KEYRING and SAFKEYRING are mutually exclusive.

Figure 12-62 is a sample DCAS configuration file that was used in the DCAS startup procedure in the next step.

```
TCPIP TCPIPB
PORT 8990
CLIENTAUTH LOCAL2
SAFKEYRING r2617.mvs28b.dcas.keyring ¶
# KEYRING /etc/dcas/dcas.kdb
# STASHFILE /etc/dcas/dcas.sth
```

Figure 12-62 DCAS configuration file

¶ The keyring name is case sensitive.

4. Start DCAS

Figure 12-63 is a sample procedure for DCAS. It is also provided in hlq.SEZAINST(EZADCASP).

```
//DCAS PROC
//DCAS EXEC PGM=EZADCDMN,REGION=4M,TIME=NOLIMIT,
// PARM=(' POSIX(ON) ALL31(ON) ',
// 'ENVAR("_CEE_ENVFILE=DD:STDENV" ',
// '"DCAS_CONFIG_FILE=/etc/dcas.r2617.conf")/-d 3') ¶
//CEEDUMP DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSERR DD SYSOUT=*
//SYSOUT DD SYSOUT=*
//STDENV DD DSN=TCPIP.TCPPARMS.R2617(DCASENV),DISP=SHR
```

Figure 12-63 DCAS procedure

¶ The DCAS configuration file is /etc/dcas.r2617.conf.

By default DCAS writes the messages in the /tmp/dcas.log file. You can set the debug level in the -d start option.

When DCAS is started as an MVS started procedure, the messages shown in Figure 12-64 on page 341 will show up in the MVS console.

```
S DCAS
$HASP100 DCAS ON STCINRDR
IEF695I START DCAS WITH JOBNAME DCAS IS ASSIGNED TO USER
TCPIP3 , GROUP OMVSGRP
$HASP373 DCAS STARTED
IEF403I DCAS - STARTED - TIME=20.36.55
EZZ8601I DCAS IS STARTING
EZZ8620I DCAS SECURITY SERVER SERVAUTH CLASS IS ACTIVE
EZZ8624I DCAS PROCESSING CONFIGURATION FILE /ETC/DCAS.R2617.CONF
EZZ8625I DCAS CONFIGURATION FILE PROCESSING IS COMPLETE
EZZ8618I DCAS LISTENING ON SECURE PORT 8990
```

Figure 12-64 DCAS startup messages

5. Define a HOD TN3270 session and create a macro for Express Logon. Refer to step 4 in 12.3.3, “Implementing ELF in a two-tier design” on page 330. The client setup is identical for two-tier and three-tier designs.

For more information about the ELF implementation with OS/390 V2R10 using the three-tier network design, refer to the following documents:

- ▶ White paper *Setting up and Using the IBM Express Logon Feature*, available at:
<http://www.ibm.com/software/network/commserver/library/>
- ▶ *IBM Communications Server for OS/390 TCP/IP 2000 Update Technical Presentation Guide*, SG24-6162.
- ▶ *Express Logon User's Guide*, available at:
<http://www.ibm.com/software/network/commserver/library/whitepapers/csos390.html>



UNIX remote execution applications

In this chapter we talk about the security considerations of the UNIX servers for remote access on z/OS.

The rlogin daemon and UNIX otelnetd allow Telnet-type access from other systems.

The rshd daemon allows a shell command to be executed from a remote client. Since the rshd protocol does not have the concept of a password, the z/OS implementation of rshd concatenates the user ID and password into the loginid parameter at the client. The rshd server on z/OS optionally supports Kerberos.

The rexecd daemon allows a shell command to be executed from a remote client. Rexecd differs from rshd in that the protocol specifically implements passwords from the client.

We show how these remote tools are used, and discuss the use of Kerberos for the rshd and otelnetd servers.

For information on configuring these daemons, see the redbook *Communications Server for z/OS V1R2 TCP/IP Implementation Guide Volume 2: UNIX Applications*, SG24-5228.

13.1 UNIX Telnet server security

The UNIX System Services Telnet server is used to enable remote Telnet clients to log on to your UNIX shell environment in either *raw* mode (also called *character-at-a-time* mode) or *line* mode.

The UNIX Telnet server is started by InetD for each incoming Telnet connection. When the Telnet session is complete, the Telnet server will exit. Each active Telnet session will have a separate instance of the Telnet server that will communicate with the Telnet client.

Figure 13-1 shows an overview of how the Telnet server is implemented in UNIX System Services.

InetD listens for Telnet connection requests on port 23 (this port number is configurable in the `/etc/inetd.conf` file). Immediately after such a request arrives, InetD clones itself by means of the `fork()` service and starts the Telnet server program as specified in `/etc/inetd.conf`.

The Telnet server program enters the initial session setup window with the Telnet client and requests a RACF user ID with a valid OMVS segment and, depending on the `otelnetd` start parameters in the InetD configuration file, a password from the client end user. In this phase user ID and password are checked, and Kerberos authentication is performed, if configured. After these are accepted by the current phase of the Telnet server, the Telnet server restarts itself via a new spawn call with a new set of parameters and, in addition, a shell process is started.

Both the Telnet server process and the shell process execute under the end user ID, and not the original user ID with `UID=0` assigned to the Telnet server.

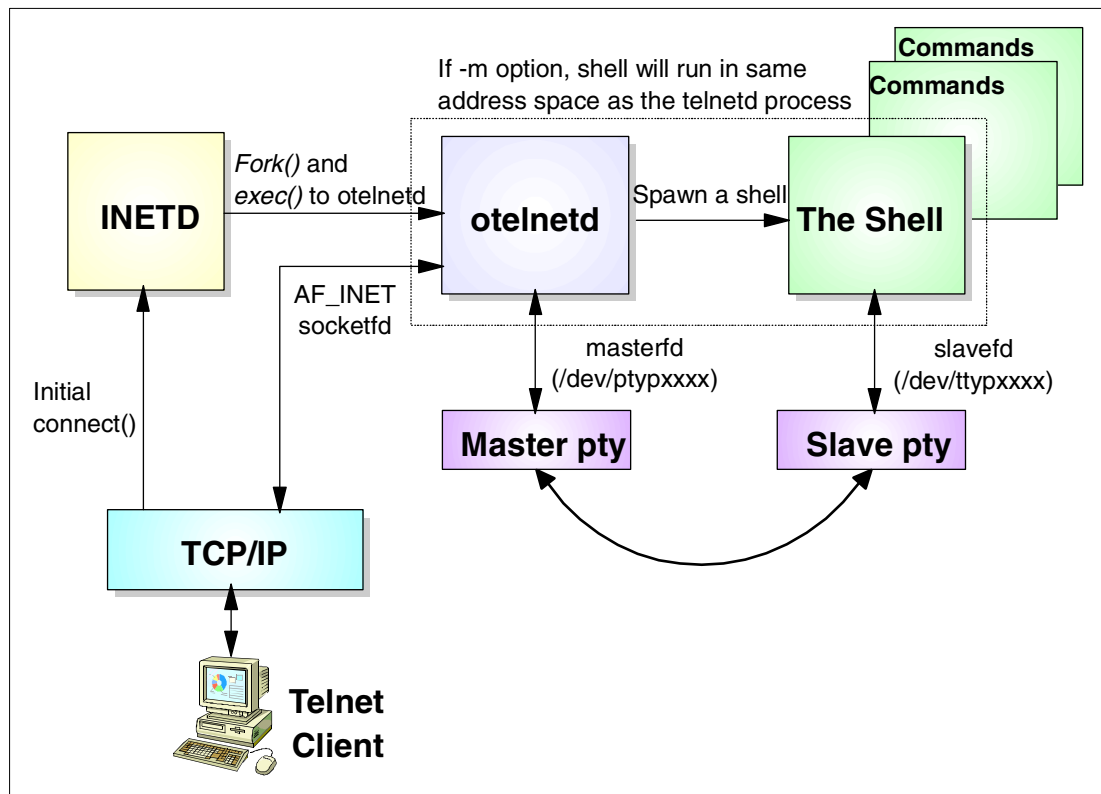


Figure 13-1 z/OS UNIX Telnet server overview

The z/OS UNIX Telnet server receives user ID and passwords from a Telnet client in clear text. This means that hackers have the possibility of eavesdropping and detecting the z/OS user ID and password from a trace. If you are not encrypting Telnet traffic with a lower-level protocol, such as IPSec, and you want to encrypt the Telnet messages, you must use Kerberos. Currently the UNIX Telnet server does not support SSL or TLS.

Tip: UNIX System Services does not provide an `/etc/passwd` file, as in other UNIX implementations. All authentications for the UNIX Telnet server are conducted by RACF, and optionally, Kerberos. If malicious hackers try to get at the `/etc/passwd` file in the HFS in order to find user IDs with strong authority such as a superuser, their attempts will end in failure.

13.1.1 Kerberized UNIX Telnet server support

Kerberos, a network authentication protocol, is designed to provide strong authentication for client/server applications using symmetric key cryptography.

UNIX Telnet became Kerberized in z/OS V1R2 to provide stronger security and allow secure data traffic in a network. This support is implemented using the Kerberos Version 5 APIs.

The user data transferred between a Telnet server and client can be protected by being encrypted with a symmetric key obtained from the Kerberos Version 5 server.

The implementation is based on the following RFCs:

- ▶ RFC 2941 - Telnet Authentication Option
- ▶ RFC 2942 - Telnet Authentication: Kerberos Version 5
- ▶ RFC 2946 - Telnet Data Encryption Option
- ▶ RFC 2952 - Telnet Encryption: DES 64 bit Cipher Feedback
- ▶ RFC 2953 - Telnet Encryption: DES 64 bit Output Feedback

For more information about the Kerberos Version 5 protocol, and a brief overview of how it is implemented on z/OS refer to 9.3, “Kerberos-based security system” on page 192.

Implementing Kerberos on otelnetd

The otelnetd daemon is started by the InetD daemon if there is an entry in the InetD configuration file for it. The default InetD configuration file is contained in an HFS file `/etc/inetd.conf`. A portion of this file can be seen in Figure 13-8 on page 350. The otelnetd line commented out with the `#` in column 1 shows how the otelnetd daemon is started without Kerberos authentication.

```
#=====
# service | socket | protocol | wait/ | user | server | server program
# name    | type  |          | nowait|     | program | arguments
#=====
#
#otelnet stream tcp nowait OMVSKERN /usr/sbin/otelnetd otelnetd -m
otelnet stream tcp nowait OMVSKERN /usr/sbin/otelnetd otelnetd -m -a valid
```

Figure 13-2 Portion of InetD configuration file showing options for otelnetd daemon

Without Kerberos, a normal RACF user ID and password are requested of the remote client. With Kerberos authentication, you can request a number of different levels depending on the '-a' switch on the **otelnetd** command:

- a valid Only allow connections when the remote user can provide valid authentication information to identify the remote user. Thus, for **otelnetd**, Kerberos authentication will be required. User verification will still occur through the login and password prompt. However, if the login user ID matches the name in the Kerberos principal, then no password will be requested. This is the most secure authentication mode.
- a user Only allow connections when the remote user can provide valid authentication information to identify the remote user, and is allowed access to the specified account without providing a password. Thus, for **otelnetd**, Kerberos authentication is required. The NAME received during AUTHENTICATION option negotiation must match the name in the Kerberos principal and be a valid user ID on the host. No user verification will occur through the login or password prompt
- a none This is the default state. Authentication information is not required. User verification will still occur through the login and password prompt. However, if the login user ID matches the name in the Kerberos principal, then no password will be requested.
- a off This disables the authentication code. All user verification happens through the login and password prompt. During option negotiation, **otelnetd** will not send DO AUTHENTICATION and, if necessary, will send DONT AUTHENTICATION.

13.2 UNIX System Services rlogind/rshd/rexecd

z/OS UNIX System Services provides safeguards to eliminate many of the typical exposures associated with TCP/IP applications. IBM has eliminated the use of one of the most dangerous features of traditional UNIX internetworking, the use of trusted host facilities.

In many UNIX systems, the system administrator may create a file, `/etc/hosts.equiv`, of trusted systems. Alternatively, each user may create a file, `.rhosts`, in the user's home directory, which lists hosts that the particular user trusts. In either case, the trusted host designation means that user IDs on other hosts may log in using `rlogin` or issue remote shell commands without specifying a password. While this feature may be convenient in some cases, often mistakes are made, which leave systems and servers open to penetration.

For example, a system administrator on `sun.tsc.ibm.com` creates a `/etc/hosts.equiv` file shown in the following:

```
aix.tsc.ibm.com    mike
```

This means that a user named `mike` on the host `aix.tsc.ibm.com` can access `sun.tsc.ibm.com` without a password. If both hosts trust each other, this file is very useful. However, if a malicious user modifies this file to the following, this host is open to the public and every user can access it without a password:

```
+
```

If users with a high level of authority access this system, they can modify, delete, and steal critical information. This is very dangerous regardless of the location of the host.

IBM has eliminated this potential problem for UNIX System Services. A password is *always* required to use `rlogin/rexec` on UNIX System Services.

Figure 13-3 shows an example of using `rlogin` (in this case, from an AIX client) to access the `rlogind` server on z/OS. Note that the RACF password is requested.

```
m10df5ff:/ >rlogin 9.12.6.30 -l focas
FOMR0226 focas's Password:
IBM
Licensed Material - Property of IBM
5694-A01 (C) Copyright IBM Corp. 1993, 2001
(C) Copyright Mortice Kern Systems, Inc., 1985, 1996.
(C) Copyright Software Development Group, University of Waterloo, 1989.

All Rights Reserved.

U.S. Government users - RESTRICTED RIGHTS - Use, Duplication, or
Disclosure restricted by GSA-ADP schedule contract with IBM Corp.

IBM is a registered trademark of the IBM Corp.

/pfocas/.profile complete
FOCAS @ SC63:/pfocas>
```

Figure 13-3 Example of using UNIX `rlogin`

Figure 13-4 shows an example of using `rexec` (in this case, from an AIX client) to access the `rexec` server on z/OS. Note that the RACF user ID and password are requested.

```
m10df5ff:/ >rexec 9.12.6.30 ls -la
Name (wtsc63oe.itso.ibm.com:root): focas
Password (wtsc63oe.itso.ibm.com:focas):
total 256
drwxr-xr-x  2 HAIMO  SYS1      8192 May 28 13:16 .
drwxr-xr-x 41 HAIMO  SYS1      8192 May 28 13:16 ..
-rwx----- 1 HAIMO  SYS1      122 May 24 18:42 .profile
-rw----- 1 HAIMO  SYS1     1750 May 28 13:20 .sh_history
-rwxr-xr-x  1 HAIMO  SYS1    77824 May 14 20:09 a.out
-rw-r--r--  1 HAIMO  SYS1     957 May 23 20:04 envvar
-rw-r--r--  1 HAIMO  SYS1     873 May 23 20:04 krb5.conf
-rw-----  1 HAIMO  SYS1      49 May 27 20:09 krb5ccname
-rwx----- 1 HAIMO  SYS1    1087 May 14 20:09 pfocas.c
-rw-r--r--  1 HAIMO  SYS1    5040 May 14 20:09 pfocas.o
-rwx----- 1 HAIMO  SYS1     149 May 14 20:22 stderr
-rwx----- 1 HAIMO  SYS1      0 May 14 20:21 stdout
m10df5ff:/ >
```

Figure 13-4 Example of using UNIX `rexec`

The `rsh` server is used, like the `rexec` server, to issue UNIX shell commands remotely. The `rsh` server does not require a password. If a user tries to use `rsh` without a password, a user exit (`ruserok`) is invoked, which you must code to ensure a suitable level of security (for example, by demanding a password). To enable the password exit for `rshd` to be invoked, you must enable the `-r` option on the `rsh` server line in the `inetd.conf` file. When the installation exit is driven, `rshd` looks for a program in `/usr/sbin` named `ruserok`. This is the only name that it will look for. If `/usr/sbin/ruserok` is not found, the request will fail.

When the `rshd` server invokes `/usr/sbin/ruserok`, it passes parameters in the following order:

1. Host name
2. Local user's UID

3. Remote user ID
4. Local user ID

If rshd receives a return code of zero from the installation exit, rshd continues. Any nonzero return code from the installation exit will cause rshd to issue the message EZYRS25E to the client and terminate all connections. The following code fragment can be used as an example to begin building a working ruserok installation exit:

```
int main(argc, argv)
int argc;
char *argv[];
char *rhost1; /* "hostname" or "hostname.domain" of client
obtained by caller:
gethostbyaddr(getpeername()) */
int cliuid; /* uid of the user name on client@s system */
char *cliuname; /* user name on client@s system */
char *servuname; /* user name on this (server@s) system */
int rc = 4;
rhost1 = argv[1];
cliuid = atoi(argv[2]);
cliuname = argv[3];
servuname = argv[4];
.
<authenticate user and set rc=0 if valid>
.
return(rc);
```

```
C:\WINNT\system32>rsh 9.12.6.30 -l focas ls
9.12.6.30: Rshd: EZYRS25E Unknown login.
rsh: can't establish connection
```

Figure 13-5 Using rsh client on Windows 2000 to execute the 'ls' UNIX command on z/OS

Figure 13-5 shows a rsh client (Windows 2000 in this case) targeting a z/OS UNIX host (9.12.6.30) with a '-l' parameter specifying RACF user ID "focas" and no password. The UNIX command being attempted is `ls`. Figure 13-6 shows the system log on the z/OS system detailing the failed attempt, since no password was supplied on the `rsh` command.

```
BPXF024I (OMVSKERN) May 28 13:58:52 rshd[83951781]: EZYRS01I MVS OE
109
RSHD BASE
BPXF024I (OMVSKERN) May 28 13:58:52 rshd[83951781]: EZYRS50E 110
__passwd() failed.
BPXF024I (OMVSKERN) May 28 13:58:52 rshd[83951781]: EZYRS45E focas
111
as focas: permission denied. cmd = 'ls'
```

Figure 13-6 Error for rsh server when no password supplied by client

Figure 13-7 on page 349 shows an rsh client (AIX in this case) accessing the z/OS rsh server at IP address 9.12.6.30 using RACF user ID "focas" with password "bermu5a". The command executed on z/OS UNIX is `ls -la`.

```

m10df5ff:/ >rsh 9.12.6.30 -l focas/bermu5a ls -la
total 256
drwxr-xr-x  2 HAIMO  SYS1      8192 May 28 13:41 .
drwxr-xr-x 41 HAIMO  SYS1      8192 May 28 13:37 ..
-rwx----- 1 HAIMO  SYS1       122 May 24 18:42 .profile
-rw----- 1 HAIMO  SYS1      1779 May 28 13:41 .sh_history
-rwxr-xr-x  1 HAIMO  SYS1     77824 May 14 20:09 a.out
-rw-r--r--  1 HAIMO  SYS1       957 May 23 20:04 envar
-rw-r--r--  1 HAIMO  SYS1       873 May 23 20:04 krb5.conf
-rw----- 1 HAIMO  SYS1        49 May 27 20:09 krb5ccname
-rwx----- 1 HAIMO  SYS1     1087 May 14 20:09 pfocas.c
-rw-r--r--  1 HAIMO  SYS1     5040 May 14 20:09 pfocas.o
-rwx----- 1 HAIMO  SYS1       149 May 14 20:22 stderr
-rwx----- 1 HAIMO  SYS1         0 May 14 20:21 stdout
m10df5ff:/ >

```

Figure 13-7 Example of using rsh client to access z/OS passing user ID and password

As discussed in this section, the z/OS rlogin/rsh/rexec daemon requires users to enter their RACF user ID and password. Because of this, the daemon can be quite secure compared to other UNIX systems, on which users can log in to these daemons without passwords. However, similar to z/OS Telnet or FTPD, user IDs and passwords that users enter are transmitted in clear text. Therefore, if you plan to use these daemons through non-secure networks such as the Internet, we recommend that you encrypt your data using IPsec. The rsh server, on the other hand, can use Kerberos to authenticate the rsh client. This is discussed next.

13.3 z/OS UNIX rshd Kerberos support

In z/OS V1R2, the z/OS UNIX rshd server can support the Kerberos Version 5 authentication technologies. Kerberos, a network authentication protocol, is designed to provide strong authentication for client/server applications using symmetric key cryptography.

Either Kerberos Version 5 authentication mechanism or that of GSS-API may be used to authenticate clients. If the user name in the Kerberos or GSS-API credentials matches the local user ID (local_userID) of the client supplied by the RSH client, then no password is required.

For more information about the Kerberos Version 5 protocol, and a brief overview of how it is implemented on z/OS refer to 9.3, “Kerberos-based security system” on page 192.

The orshd daemon is started by the InetD daemon if there is an entry in the InetD configuration file for it. The default InetD configuration file is contained in an HFS file ‘/etc/inetd.conf’. A portion of this file can be seen in Figure 13-8 on page 350. The orshd line commented out with the ‘#’ in column 1 shows how the orshd daemon is started without Kerberos authentication.

```

#=====
# service | socket | protocol | wait/ | user | server | server program
# name   | type  |         | nowait|     | program | arguments
#=====
#
shell   stream tcp nowait OMVSKERN /usr/sbin/orshd orshd -LV

```

Figure 13-8 Portion of InetD configuration file showing options for orshd daemon

13.3.1 Implementing Kerberos on orshd

To start the rsh server with Kerberos, there are some new command line options that need to be specified in the inetd.conf file.

```

#=====
# service | socket | protocol | wait/ | user | server | server program
# name   | type  |         | nowait|     | program | arguments
#=====
#
shell   stream tcp nowait OMVSKERN /usr/sbin/orshd orshd -k KRB5 -e -m

```

Figure 13-9 Portion of InetD configuration file showing Kerberos options for orshd daemon

Figure 13-9 shows the line needed in the inetd.conf file to specify Kerberos options for the rsh server. These command line options are discussed below.

Without Kerberos, a normal RACF user ID and password are requested of the remote client. With Kerberos authentication, you can request a number of different levels depending on the '-a' switch on the **orshd** command:

- k mechanism Specifies the authentication mechanism to be used to authenticate the client. Valid values for mechanism are KRB5 and GSSAPI.
- e Require the client to encrypt the connection.
- m Require Kerberos5 clients to present a cryptographic checksum of initial connection information like the name of the user that the client is trying to access in the initial authenticator. This checksum provides additional security by preventing an attacker from changing the initial connection information. If this option is specified, older Kerberos5 clients that do not send a checksum in the authenticator will not be able to authenticate to this server. This option is mutually exclusive with the -i option and is only valid if -a KRB5 is specified.

If neither the -c or -i options are specified, then checksums are validated if presented. Since it is difficult to remove a checksum from an authenticator without making the authenticator invalid, this default mode is almost as significant of a security improvement as -c if new clients are used. It has the additional advantage of backwards compatability with some clients. Unfortunately, clients before Kerberos V5, Beta5, generate invalid checksums; if these clients are used, the -i option must be used.
- i Ignore authenticator checksums if provided. This option ignores authenticator checksums presented by current Kerberos clients to protect initial connection information; it is the opposite of -c. This option is provided because some older clients, particularly clients predating the release of Kerberos V5 Beta5 (May 1995), present invalid checksums that prevent Kerberos authentication

from succeeding in the default mode. This option is mutually exclusive with the -c option and is only valid if -a KRB5 is specified.

-t

This option may be used to set the KRB5_SERVER_KEYTAB environment variable. If this environment variable is set, the Security Runtime will use a local instance of the Kerberos security server to decrypt service tickets instead of obtaining the key from a key table. The orshd application must have at least READ access to the IRR.RUSERMAP facility in order to use this capability. For more information see *z/OS V1R2.0 SecureWay Security Server Network Authentication Server Administration*, SC24-5926.



OMPRoute security

Establishing the routing tables in an IP network is critical to successful communication across that network. Many smaller network configurations rely strictly on static routing definitions to maintain the routing infrastructure. As with other manual processes, this method is prone to errors and is time-consuming. As a result, when the management of the network routing tables becomes too labor-intensive, a more efficient and reliable means of dealing with the network routes is desirable. This is where dynamic routing protocols play a significant role.

With dynamic routing protocols, changes to the routes occur through dynamic routing updates. They are essentially transparent to the network administrator since the updates are exchanged among the routers participating in the network. With the proper access to the physical infrastructure and wiring of an IP network, it would be possible for someone to introduce an unauthorized router into the network if dynamic routing protocols were in effect. The “rogue” router could then either interconnect the existing network with an untrusted network over which the dynamic routing updates would continue to flow, or could introduce routing updates that are disruptive to the normal operation of the network.

In this chapter, we talk about the security considerations for OMPRoute, and in particular, the OSPF protocol. OSPF has always had a password verification scheme for route update messages, but they were sent “in the clear” (no encryption).

OSPF now has the facility to append a message digest to route updates, and to use a shared password to encrypt that digest to produce a message authentication code (MAC).

Note: z/OS V1R3 will probably be the last version that supports the dynamic routing daemon OROUTED. OMPROUTE supports all protocols that OROUTED does, therefore making OROUTED obsolete. To make migration from OROUTED easier, a new parameter switch (“-c”) has been added to OROUTED to produce an OMPROUTE configuration from the OROUTED one. Consequently, OROUTED is not discussed here.

14.1 OSPF route update messages security

OSPF authentication is a means of verifying that OSPF packets are genuine. There are two methods of OSPF authentication: password, and MD5 cryptographic. Password authentication is very basic: an 8-byte password is appended to all OSPF packets and sent in the clear with the rest of the packet. If the sent password matches that defined by the packet receiver, the packet is accepted. MD5 authentication is more sophisticated. The combination of the OSPF packet and the MD5 key is summarized into a 16-byte message digest, which is appended to the packet and sent. The keys are never sent, only the message digests. The receiver then attempts to recreate the message digest from the combination of its defined key and the OSPF packet. If the digest is successfully recreated, the packet is accepted; otherwise it is rejected.

Note: Without MD5 authentication, the clear-text password is not a foolproof shield against a malicious router's intrusion into a network. It may be regarded only as a safeguard against the accidental introduction of an invalid routing node into your networking infrastructure.

14.2 OMPRoute configuration

This section describes OMPRoute configuration.

14.2.1 The Area configuration statement

OMPROUTE designates the default OSPF area authentication scheme in the Area statement. This authentication type can be set to None, MD5 or Password. The actual key (for the MD5 or Password schemes) is specified on the OSPF_INTERFACE statement(s). If the Authentication_type is set to Password, the actual password will traverse the network in the clear (unencrypted). This is not recommended and should only be used if the other OSPF routers in the network cannot support the MD5 authentication. For an example of how to set the default authentication across all OSPF interfaces to MD5, see Figure 14-1.

```
Area
  Area_Number=1.1.1.1
  Authentication_Type=MD5;
;
```

Figure 14-1 OMPRoute configuration showing authentication scheme MD5 on Area statement

If you are using MD5 authentication, you must enter a 16-byte hex value for the key on all OSPF_INTERFACE statements that will use it. You can make up a password and distribute it to other OSPF routers that you will be communicating with, or you can use the z/OS UNIX utility **pwtkey** to generate the 16-byte hex value from a given character password.

For information on how to use **pwtkey**, see *z/OS V1R2.0 CS: IP System Administrator's Commands*, SC31-8781.

Figure 14-2 on page 355 shows how a 16-byte MD5 key is generated from the password "passexample".

```

FOCAS @ SC63:/pfocas>pwtokey -p HMAC-MD5 -u auth 'passexample'
Display of 16 byte HMAC-MD5 authKey:
    c294d23d50954dcbf21ed7ef0858c18a

FOCAS @ SC63:/pfocas>

```

Figure 14-2 Generating an MD5 key value for the OSPF interfaces

14.2.2 The OSPF_Interface configuration statement

Individual OSPF interfaces can override the default security scheme by specifying `Authentication_Type` on the `OSPF_INTERFACE` (or `VIRTUAL_LINK`) statement. If `Authentication_Type` is not coded, it is defaulted from the Area statement. Continuing our example of MD5 authentication, we take the 16 byte hex value generated from `pwtokey` in Figure 14-2 and paste it into the `OSPF_INTERFACE(s)` that will be using this authentication scheme with other OSPF routers.

```

OSPF_Interface
  IP_Address=9.59.101.5
  Authentication_Type=MD5
  Authentication_Key=0xc294d23d50954dcbf21ed7ef0858c18a
  Name=TR1
  Subnet_Mask=255.255.255.0
  Attaches_To_Area=0.0.0.0
  Cost0=2
  Router_Priority=1;
;

```

Figure 14-3 Example of an OSPF interface using an MD5 authentication scheme

The `Authentication_Key` value in Figure 14-3 would also have to be distributed to all OSPF routers that this interface will communicate with. Note the `Authentication_Type` statement did not need to be coded on the `OSPF_Interface` statement if it was already specified on the Area statement.

The `Authentication_Key` value in Figure 14-3 would also have to be distributed to all OSPF routers that this interface will communicate with.

```

OSPF_Interface
  IP_Address=9.59.101.5
  Authentication_Type=Password
  Authentication_Key="mypasswd"
  Name=TR1
  Subnet_Mask=255.255.255.0
  Attaches_To_Area=0.0.0.0
  Cost0=2
  Router_Priority=1;
;

```

Figure 14-4 Example of an OSPF interface using a password authentication scheme

If the Area statement had `Authentication_Type=MD5` specified, but you wanted to use the simple password authentication scheme on a particular interface, you would override the Authentication Type on the `OSPF_INTERFACE` statement. Figure 14-4 on page 355 shows an example of an interface using the simple password scheme. The password specified is “mypasswd”, which is limited to 8 characters or less, and must be distributed to all OSPF routers that this interface will communicate with.

For information relating to the implementation and syntax of these security features in OMPROUTE, please consult the *z/OS Communications Server IP Configuration Guide*, SC31-8775 and *z/OS Communications Server IP Configuration Reference*, SC31-8776.

For a short tutorial on OSPF, please see *Communications Server for z/OS V1R2 TCP/IP Implementation Guide, Volume 4: Connectivity and Routing*, SG24-6516.



Network management applications

In this chapter we talk about the security considerations of the network management protocols. We cover the following applications:

- ▶ z/OS SNMP
- ▶ z/OS Policy Agent

15.1 z/OS SNMP

Simple Network Management Protocol (SNMP) defines an architecture that consists of:

- ▶ Network management applications
- ▶ Network management agents and subagents
- ▶ Network elements or objects

The SNMP network management application can ask agents for specific information about network elements. Conversely, agents can notify a network management application when something happens to one or more network elements. The protocol used between the network management application and agents is SNMP.

The transport protocol for SNMP requests is the User Datagram Protocol (UDP).

SNMP defines both the network management data and the ways in which the data is retrieved or changed by the network management application. Examples of network management data include device definitions, counts of packets received at the IP layer and TCP connection data. The information about the network elements is stored in a Management Information Base (MIB), which is supported by the SNMP agent and its subagents. A MIB variable (or MIB object) is a specific instance of data in a collection of objects related to a common management area.

Figure 15-1 illustrates the relationships between manager, agents, and subagents.

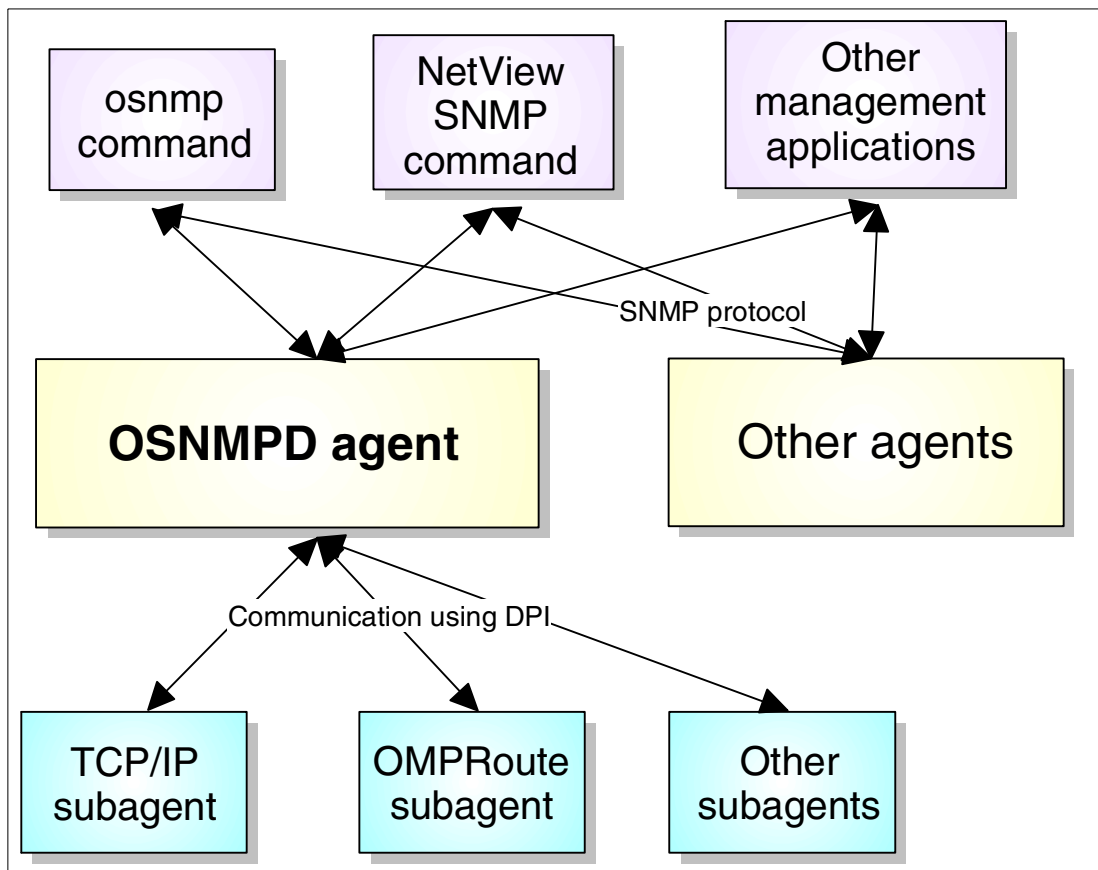


Figure 15-1 Functional components of SNMP

The following list illustrates the sequence of events from the time you issue an SNMP command until you receive the response:

1. The user issues a NetView SNMP (SNMP) or z/OS SNMP (`osnmp`) command.
2. The command processor validates and encodes the request in a protocol data unit (PDU), and sends it to the SNMP agent.
3. The SNMP agent validates the request and, if necessary, sends it to an SNMP subagent (for example, the OMPRoute subagent, see Figure 15-1 on page 358). Requests for agent-oriented objects are handled by the agent and all others are handled by a subagent. To determine which objects are handled by the agent and which by a subagent, refer to *z/OS V1R2.0 CS: IP User's Guide and Commands*, SC31-8780.
4. The agent sends the response to the originator of the request. The command processor displays the response.

15.1.1 SNMP security

Perhaps you are wondering, why security at all? All a hacker can do is read a bunch of variables and maybe change a few minor ones. Ah, but if your hacker reads through some of the MIB variables, he or she might find some very interesting read/write objects. The MIB object `ipForwarding`, for example, is a read/write variable. If `ipForwarding` were to be changed from 1 (routing) to 2 (not routing), what would be the consequences to your network? In addition, the potential hacker could change or generate trap or inform values, causing your SNMP manager to make an incorrect decision. Like the rest of the TCP/IP world, SNMP applications want to be able to know who the data came from and whether the data may have been modified.

The SNMP agent supports what amounts to two types of security:

- ▶ SNMPv1 and SNMPv2c
- ▶ SNMPv3 security

SNMPv1 and SNMPv2c are community-based security, where a community name (or password) is passed with a request. If the community name is recognized as one that can be used by the IP address from which the request originates, the SNMP agent processes the request. SNMPv1 and SNMPv2c have apparent limitations: no encryption is used, no user IDs are required. Without user IDs, no limit can be placed on the scope of MIB objects that can be queried or set.

Note: The experimental SNMPv2u security is no longer supported. It has been replaced by the standards-based SNMPv3 network management framework. SNMPv3 was introduced in OS/390 V2R7.

SNMPv3 provides a more powerful and flexible framework for message security and access control. Message security involves providing:

- ▶ Data integrity checking, to ensure that the data was not altered in transit.
- ▶ Data origin verification, to ensure that the request or response originates from the source from which it claims to have come.
- ▶ Message timeliness checking and, optionally, data confidentiality, to protect against eavesdropping. Access control is the ability to control exactly what data an individual user can read or write.

The SNMPv3 architecture introduces the User-Based Security Model (USM) for message security and the View-Based Access Control Model (VACM) for access control. The architecture supports the concurrent use of different security, access control, and message processing models. For example, community-based security can be used concurrently with USM (they do not interact with each other; they can simply coexist without interference).

USM

USM uses the concept of a user for which security parameters (levels of security, authentication and privacy protocols, and keys) are configured at both the agent and the manager. Messages sent using USM are better protected than messages sent with community-based security, where passwords are sent in the clear and can be displayed in traces. With USM, messages exchanged between the manager and the agent have data integrity checking and data origin authentication. Message delays and message replays (beyond what happens normally due to a connectionless transport protocol) are protected against with the use of time indicators and request IDs.

USM supports two authentication algorithms: HMAC-SHA and HMAC-MD5. The generation of the authentication keys involves some manual intervention on behalf of the system administrator. Things are simplified somewhat by the `pwtkey` utility. This utility allows a password to be used to generate a key.

For data confidentiality, symmetric encryption algorithms are used. No asymmetric encryption is available. CBC 56-bit DES is the only encryption available. Again, generation of the symmetric key can be done using the `pwtkey` facility with a password as input.

Note that the password used in generation of these keys is no relation to the community name (password) used for pre-SNMPv3 protocols. The password is simply a method of simplifying the generation of authentication and privacy keys.

For SNMP agents and managers, these keys must be stored into configuration files (`snmpd.conf` and `osnmp.conf` on the OS/390 platform, respectively). Protection of these configuration files is paramount to maintain the security of your SNMP environment.

Both the authentication and encryption keys must be manually configured at each SNMP agent or manager. This isn't as great a difficulty as it appears, since there should be very few locations on your network that require SNMP interactions.

VACM

The use of VACM involves defining collections of data (called views), groups of users of the data, and access statements that define which views a particular group of users can use for reading, writing, or receipt in a trap. This is a powerful and important tool.

VACM views can be used to limit the range of MIB objects that a SNMP management application can access. From there, the view can be restricted to read or write privileges. The view can also be restricted on whether traps or informs can be sent to a given user.

See Figure 15-2 on page 361 for more information on USM and VCAM.

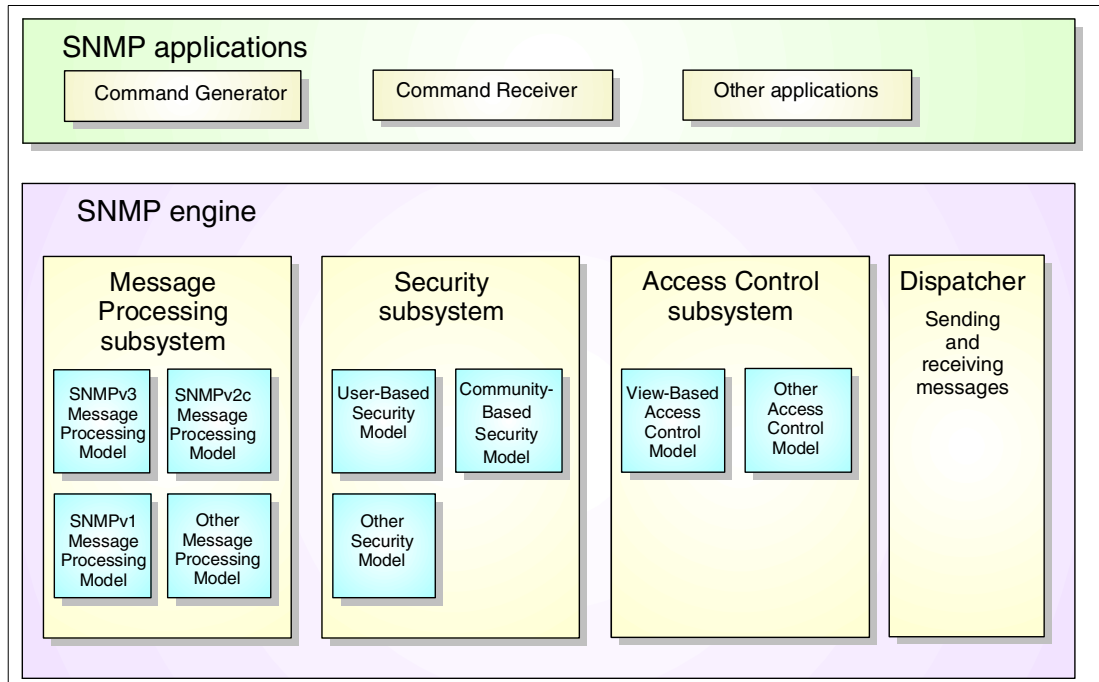


Figure 15-2 SNMP subsystem processes

Dynamic changes

SNMPv3 also introduces the ability to configure dynamically the SNMP agent using SNMP SET commands against the MIB objects that represent the agent's configuration. This dynamic configuration support enables addition, deletion, and modification of configuration entries either locally or remotely.

Remote modification of user keys can be especially useful, simplifying the manual configuration process. The `pwchange` utility can be used to generate new `keyChange` value. Use of the `keyChange` value enables keys to be changed over the wire without actually sending any keys.

In summary

From the above discussion it is apparent that SNMPv3 provides a reasonable level of security. However, SNMPv3 is not supported by as wide a range of platforms as the older versions. If your network includes SNMP hosts capable of only V1 or V2, then we recommend two solutions:

- ▶ Use IPsec to protect SNMP traffic if you are concerned about its integrity and authenticity.
- ▶ Use VACM for community-based requests.

USM can be used only when both the SNMP agent and the manager requesting the data support USM, as do the CS for z/OS SNMP agent and the `osnmp` command. VACM can be used even for community-based requests, but doing so requires migration of existing community name and trap destination definitions. Table 15-1 on page 362 is a list of the advantages and disadvantages of using each type of security.

Table 15-1 SNMP security overview

SNMPv1/SNMPv2c Advantages	SNMPv3 Disadvantages
Traditional standards-based administrative model	Emerging standards-based administrative model
Easy to configure	More involved configuration options
SNMPv1 and SNMPv2c allow particular IP addresses to access all data or no data	SNMPv3 allows a particular user to access particular data
Not very robust (password sent in PDU)	Robust (data integrity and data origin authentication)
Any user that can read data can also change data (for objects defined as read-write)	The ability to change data can be limited to specific users
No data confidentiality	Encryption available (separate product)
Configuration changes require restarting of SNMP agent	Configuration changes for USM and VACM can be made dynamically, either locally or remotely

For detailed information regarding implementation of OS/390 SNMP, see *Managing OS/390 TCP/IP with SNMP*, SG24-5866.

15.2 z/OS Policy Agent

OS/390 SecureWay Communications Server Release 7 introduced a policy agent (PAGENT) that determines and applies the level of service to be given to various traffic types. Much of its work is concerned with prioritizing IP traffic and managing resource reservations, but one of its functions is to apply access control. In this respect, it can act in a similar fashion to a firewall. Policies can block unwanted datagrams or control access for connection requests from clients. Policy Agent can get policies from either an MVS data set or an LDAP server. If an LDAP server is used, communications between Policy Agent and LDAP can be secured with SSL.

For details on configuring and using z/OS Policy Agent, see *Communications Server for z/OS V1R2 TCP/IP Implementation Guide Volume 6: Policy and Network Management*, SG24-6839. This section will be limited to configuring Policy Agent to use SSL for securing communications with the LDAP server.

15.2.1 SSL with LDAP and Policy Agent

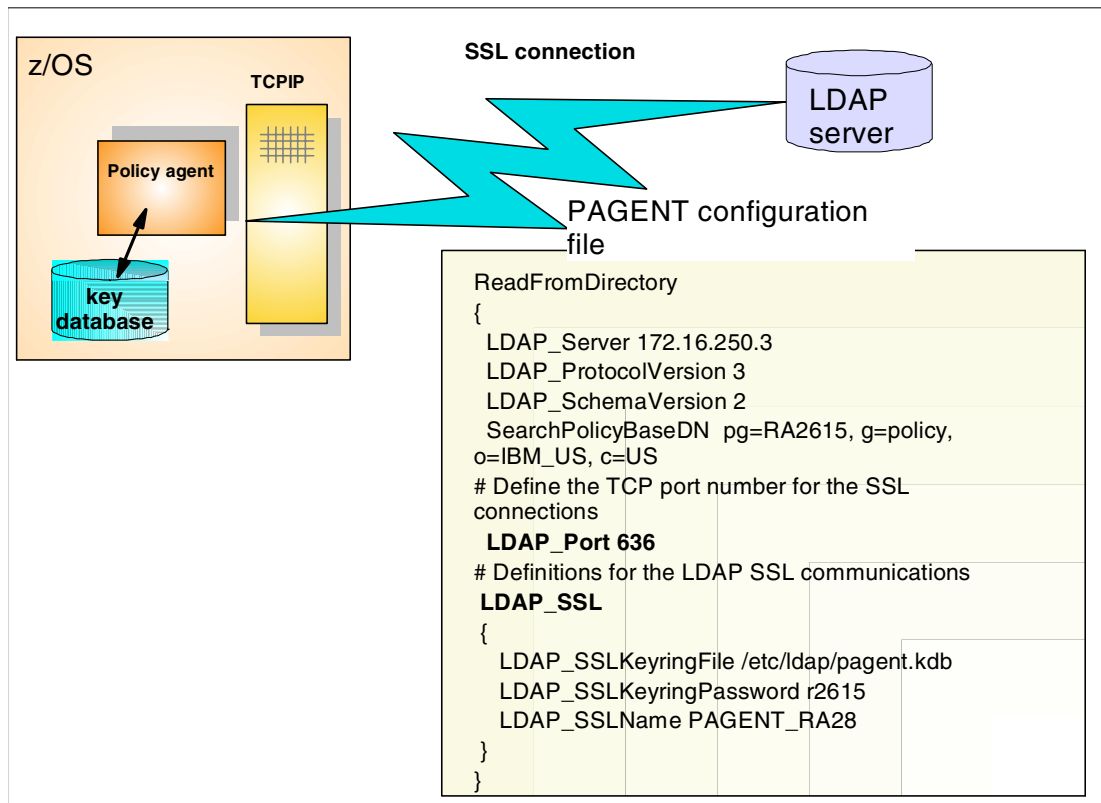


Figure 15-3 SSL connection to LDAP server

Figure 15-3 shows a portion of the z/OS PAGENT. The ReadFromDirectory statement initializes the Policy Agent as an LDAP client, and defines the location of the LDAP server, the port number to use, and the SSL parameters to use between PAGENT and LDAP. The SSL parameters specify the location of the digital certificate(s) to be used by SSL.

For general information on digital certificates, see 2.2.4, “Digital certificates” on page 13, and for a detailed discussion on how to obtain and store certificates, see Chapter 10, “Certificate management in z/OS” on page 203.

15.2.2 Considerations when opening an SSL connection

In order for the LDAP client to communicate with an LDAP server over an SSL-protected TCP/IP socket connection, the LDAP server must transmit a certificate to the LDAP client and, optionally, the client can transmit its certificate to the LDAP server. The LDAP client and server must verify that the certificates they receive are valid. Once the LDAP client and server have determined the validity of the certificates provided to them, SSL-protected communications are established between them.

You can create the certificates for your organization using self-signed certificates, and in this situation you are the Certificate Authority. Or you can obtain certificates signed by an external CA, such as VeriSign.

When you create your self-signed certificate or you receive the certificate from a CA, you must store the certificate in a keyring database. This should be a RACF database or an HFS file created with the gskkyman utility.

We defined the LDAP server to allow the LDAP client to validate the LDAP server and the LDAP server to validate the LDAP client if the client sends its digital certificate on the initial contact between the client and the server. And we used HFS keyring databases to store the self-signed LDAP server's certificate and the PAGENT certificate.

To accomplish this, follow these steps:

1. Define a keyring database for the LDAP server, as an HFS file using the `gskkyman` utility.
2. Add a self-signed certificate for the LDAP server in the keyring database.
3. In the LDAP configuration file, add the following statements for the LDAP server to support the SSL communications.

```

port 389
secureport 636 1
# security <sslonly|ssl|none>
security ssl 2
# sslAuth <serverClientAuth|serverAuth>
sslAuth serverClientAuth 3
sslKeyRingFile /etc/ldap/ldapring.kdb 4
sslKeyRingPWStashFile /etc/ldap/ldapring.sth 5
sslKeyRingFilePW r2615 6

```

We used the TCP port 636 **(1)** for secure connections, enabled SSL **(2)**, and defined the security method **(3)**, which indicates both client and server authentication are supported. In addition to those parameters above, the keyring database **(4)**, stash file names **(5)** and the password **(6)** to access to the database have to be defined.

4. Define a keyring database used by the Policy Agent. We created an HFS file using the `gskkyman` utility.
5. In the PAGENT configuration file, configure the following statements so that PAGENT uses SSL to communicate with LDAP.

```

# Define the TCP port number for the SSL connections
LDAP_Port 636 7
# Definitions for the LDAP SSL communications
LDAP_SSL 8
{
    LDAP_SSLKeyringFile /etc/ldap/pagent.kdb 9
    LDAP_SSLKeyringPassword r2615 10
    # LDAP_SSLName PAGENT_RA28 11
}

```

The port parameter **(7)** is not directly related to an SSL connection, but we had to specify the port number used to open the SSL connections to the well-known port 398. The `LDAP_SSL` **(8)** statement enables the SSL communication between PAGENT and the LDAP server, and has to be enclosed by `{ }`. The following parameters related to the SSL communication are supported:

- `LDAP_SSLKeyringFile` **(9)**, which defines the keyring file that contains the certificates of the Certificate Authorities that are trusted by the client. When client authentication is required, you must have a public key and the associated certificate stored in the key database.
- `LDAP_SSLKeyringPassword` **(10)**, which specifies the password to access the keyring file.
- `LDAP_SSLName`, which specifies the label assigned when creating your private key/certificate pair with `gskkyman` or `RACDCERT`. When the client is authenticated by the LDAP server, this parameter is required. The value is case sensitive.

6. Export the LDAP server self-signed certificate from the LDAP key database into an HFS file.
7. Store the LDAP self-signed certificate into the PAGENT keyring database.

When client authentication is required, do the following actions additionally. Note that we used a self-signed certificate also for PAGENT.

- ▶ Define a self-signed certificate in the PAGENT keyring database.
- ▶ Export the PAGENT self-signed certificate from the PAGENT keyring database, and save it in an HFS file.
- ▶ Import the PAGENT self-signed certificate into the LDAP keyring database.
- ▶ Configure the LDAP_SSLName (1) parameter in the PAGENT configuration file to let the Policy Agent send its certificate. This is the label name you define for your self-signed certificate.

Note: The LDAP_SSLName value must be defined without blanks inside.

The SSL support is now provided by the System Secure Sockets Layer (System SSL) element of z/OS and it includes a database utility called gskkyman used to create the keyring file. See *z/OS System Secure Socket Layer Programming*, SC24-5901.

For more information about the LDAP server configuration for the SSL communication, consult *LDAP Server Administration and Use*, SC24-5923.



HTTP Server for z/OS

IBM made its first World Wide Web server (Web server) for MVS/ESA available in December 1995. That product was the IBM Internet Connection Server for MVS/ESA. Before becoming IBM HTTP Server, it was also called the Domino Go Webserver.

There are still people who cannot believe that S/390 systems could run Web servers, but anybody who takes a more thorough look at the matter should quickly discover the virtues of Web serving with S/390 systems. For most installations, there is significant value in putting Web servers on z/OS. Your S/390 already holds significant amounts of your business data, which could be made available to the people who need it by using Internet technologies. Furthermore, since z/OS is a critical business system for so many enterprises, it has built-in strengths such as scalability, availability, security, and integrity. These strengths can be applied to Internet work just as they are applied today to transaction processing, very large databases, and batch processing.

16.1 HTTP Server security

In general, Web server security consists of the following elements:

- ▶ Identification and authentication

Authentication is the process of verifying the validity of a claimed identity.

- ▶ Access control

Assurance that each user or computer that uses the service is permitted to do what the user asks. The term authorization is often used as a synonym for access control, but authorization also includes granting the accessor the right to perform some actions based on access rights.

- ▶ Data confidentiality

Sensitive information must not be revealed to parties for which it was not meant. Data confidentiality is often also referred to as privacy.

- ▶ Data integrity

Data integrity ensures that the data is not altered, rearranged, or truncated in an unauthorized manner.

- ▶ Nonrepudiation

Assurance that a sender cannot deny being the source of a message and that the recipient cannot deny the receipt of a message.

- ▶ Availability

The availability over time, or in terms of response time, of a system. While this is normally not a security-related term, it may become so when dealing with denial-of-service attacks that may intentionally slow down or completely paralyze a system.

- ▶ Security audit

A process where an independent party checks the security level of an organization or computer system.

- ▶ Key management

Key management deals with the secure generation, distribution, and storage of keys used in cryptography.

IBM HTTP Server for z/OS provides functions corresponding to all of the above security elements. See Table 16-1 for a summary.

Table 16-1 Security functions of IBM HTTP Server for z/OS

Web security element	Corresponding functions of IBM HTTP Server for z/OS
Authentication	RACF user ID, password file, client certificate, LDAP definition
Access Control	Protection/Protect/DefProt directive, ACL file, LDAP information
Data Confidentiality	SSL
Data Integrity	SSL
Nonrepudiation	SSL, Association of client certificate with RACF user ID
Availability	Scalable Web server with WLM
Security Audit	Various log files and reporting tools, SNMP subagent
Key Management	gskkyman utility or RACF keyring

In this redbook, we focus on the following security topics related to the IBM HTTP Server for z/OS:

- ▶ How to authenticate Web clients
- ▶ How to control access from clients
- ▶ How to make HTTP communications secure

16.2 Server security structure

IBM HTTP Server for z/OS creates a worker thread within its process (address space) for each client's request. IBM HTTP Server can associate each thread with the identity of the user using a proprietary `pthread_security_np()` service. Other UNIX systems do not have this concept of thread security.

The `pthread_security_np()` service enables IBM HTTP Server to establish a more strict and yet more dynamic security environment. A RACF user ID that a client inputs, or a surrogate user ID that is defined in the IBM HTTP Server configuration file (`/etc/httpd.conf`) in advance, is assigned to each worker thread.

If your installation needs to control sensitive information served by the HTTP Server, it is advisable that you force browsers to enter a user ID and password. However, to force all clients on the intranet or Internet to input their user IDs and passwords whenever they access public information is not only uncommon, it is tedious to the end users. In such a case, it is better to use a surrogate user ID, allowing the HTTP Server to bring up public access files without requiring the user to enter a user ID.

16.3 Setting up SAF control

In the Web server environment, we define a resource as anything that can be delivered (served) by the Web server. For example, Web pages and CGI programs are both resources. When we define a surrogate user ID for accessing some resources, the RACF password is not specified on the `pthread_security_np()` service invocation. However, an additional check is made to ensure that the server is authorized to act as the client. UNIX System Services uses profiles defined to the RACF SURROGAT class to authorize the Web server to act as a surrogate of a client. Profiles defined to the SURROGAT class are of the form:

```
BPX.SRV.<userid>
```

in which `<userid>` is the RACF user ID of the user on behalf of whom the server will act as a surrogate.

The first step is to create a BPX.SERVER profile in class FACILITY. BPX.SERVER restricts the use of the `pthread_security_np()` service. Then, give the Web server's user ID (normally WEBSRV) READ authority to BPX.SERVER in order to authorize a server to act as a surrogate for client user IDs. The following is an example of the commands:

```
RDEFINE FACILITY BPX.SERVER UACC(NONE)
PERMIT BPX.SERVER CLASS(FACILITY) ID(WEBSRV) ACCESS(READ)
SETROPTS RACLIST(FACILITY) REFRESH
```

For more detailed information regarding BPX.SERVER, see "BPX.SERVER authority" on page 35.

You should also confirm that the server and language environment load libraries have program control turned on (with NOPADCHK). To check this, use the following RACF command:

```
RLIST PROGRAM *
```

For the IBM HTTP Server to process requests for thread-level security without passwords, follow the steps shown below. The steps below are for IBM HTTP Server with RACF user ID WEBSRV with the ability to support user ID ANONYMO without a password.

1. First, you must activate the SURROGAT class in RACF:

```
SETROPTS CLASSACT(SURROGAT)
```

This has to be done only once on the system. The SURROGAT class may already have been set up on your system. If a daemon or server you are running will be using the SURROGAT support heavily, consider using the RACLIST command to keep the SURROGAT profiles in storage. The following example shows how to cache the SURROGAT profiles in storage:

```
SETROPTS RACLIST(SURROGAT)
```

2. If the SURROGAT class has been made resident using SETROPTS RACLIST, any changes to the SURROGAT profiles must be followed by a refresh of the in-storage tables. To create the SURROGAT class profile for user ANONYMO, issue:

```
RDEFINE SURROGAT BPX.SRV.ANONYMO UACC(NONE)  
SETROPTS RACLIST(SURROGAT) REFRESH
```

A similar SURROGAT profile is required for each user ID that a server uses without specifying a password.

3. To permit server WEBSRV to create a thread-level security environment for user ANONYMO, issue the PERMIT command:

```
PERMIT BPX.SRV.ANONYMO CLASS(SURROGAT) ID(WEBSRV) ACCESS(READ)  
SETROPTS RACLIST(SURROGAT) REFRESH
```

For more information regarding thread-level security, see *z/OS V1R2.0 UNIX System Services Planning*, GA22-7800.

16.4 How to protect resources

HTTP provides an authentication mechanism to allow servers to define access permissions for clients and the resources they want to access. The authentication method can be one of the following:

- ▶ Basic authentication

Basic authentication is based on a user ID and password. In this authentication scheme, the server will permit the connection only if the user ID and password are validated. In basic authentication, user IDs and passwords are not encrypted. They are encoded in Base64 format.

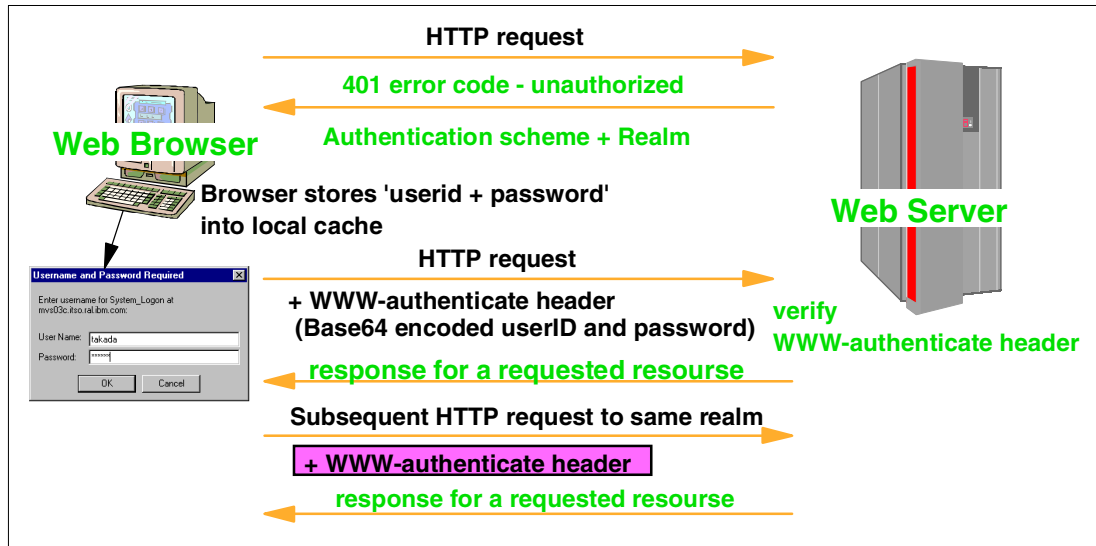


Figure 16-1 Overview of basic authentication

When the browser requests a protected resource, the Web server returns a 401 error code (unauthorized) and includes the WWW-authenticate header.

The WWW-authenticate header tells the browser which authentication schemes the Web server supports and the name of the realm that contains the protected resource. If the user has not already entered a valid user ID and password for this Web server and realm, the password window is presented (see Figure 16-1). In subsequent requests, the browser can find the information from its internal cache or password file. The browser then requests the same document, but this time with an WWW-authenticate header that includes the user's ID and password. Thus, each *logon* (supply of user ID and password) is applied to multiple requests for the same Web server and realm. The password and user ID normally remain available until the browser is recycled or its memory cache is flushed.

► HTTP digest access authentication

The digest authentication scheme is an extension of HTTP and described in RFC 2617. In this authentication scheme, the user ID and a digest containing an encrypted form of the password are sent to the server. The server computes a similar digest and grants access to the protected resources if the two digests are equal. Notice that if the digest authentication is enabled, what is sent over the network is not simply an encrypted form of the password, which could be decrypted if you had the correct key, but is a one-way hash value of the password, which cannot be decrypted. So digest authentication provides a higher level of security than the Base64 encoded password. Currently, IBM HTTP Server for z/OS does not support digest authentication. However, digest authentication is not supported by many Web browsers and other Web servers, either.

The remainder of this section assumes basic authentication is being used with the IBM HTTP Server for z/OS.

16.4.1 Access control directives

Access control directives specify the user name the server changes to before accessing files. It can be set globally (default setting) via the UserID directive. It can also be set locally (for a specific realm) within a Protection, Protect or DefProt setup. The server does not normally serve data from its own WEBSRV user ID because of its level of authority (normally, WEBSRV has UID=0).

The ability to change to a different z/OS UNIX identity is controlled on z/OS via the `pthread_security_np()` service.

The server user ID, usually `WEBSRV`, must be given surrogate authority for any specified surrogate user ID. It also needs to have `READ` access to profile `BPX.SERVER`. In `SYS1.SAMPLIB`, member `IMWJSEC` can be used to automate the creation of the security environment.

IBM HTTP Server uses four types of access control user IDs: `%%CLIENT%%`, `%%SERVER%%`, `%%CERTIF%%`, and surrogate user ID.

▶ `%%CLIENT%%`

This special string tells the server to require that the requester have a local z/OS user ID and password. The requester's user ID is used to access the data. The user is prompted for a valid password.

▶ `%%SERVER%%`

This special string tells the server to use its own user ID to access data.

Note: Be extremely cautious when using `%%SERVER%%`. Because your server is normally running as a superuser, all users have superuser authority.

Note, however, that the HTTP Server is no longer required to run under the superuser authority. Refer to APAR PQ41777 for the complete information.

▶ `%%CERTIF%%`

This special string tells the server to treat SSL connection certificate data in a special way. The Web server, when presented with an SSL session with client certificate data present, calls RACF to map the client certificate to a local z/OS user ID and password. If the session is not using SSL, there is no certificate present, the underlying support is not available, or the certificate cannot be mapped, then the request is treated as if `%%CLIENT%%` had been specified.

▶ Surrogate user ID

As mentioned in 16.2, "Server security structure" on page 369, this is generally the user ID for public access. If this user ID is specified on a `UserID` directive, no security checking is performed. All realms are accessible unless specifically protected via a `DefProt` or `Protect` directive.

For example, if you specify `ANONYMO` as a surrogate user ID in the `UserID` directive, you have to define a RACF `SURROGAT` class profile, `BPX.SRV.ANONYMO`. Then you also have to give `READ` access authority for this profile to the Web server's user ID (normally `WEBSRV`):

```
ADDGROUP EXTERNAL OMVS(GID(999))
ADDUSER ANONYMO DFLTGRP(EXTERNAL) OMVS(UID(5) HOME('/') PROG('/bin/sh'))
RDEFINE SURROGAT BPX.SRV.ANONYMO UACC(NONE)
```

```
PERMIT BPX.SRV.ANONYMO CLASS(SURROGAT) ID(WEBSRV) ACCESS(READ)
SETROPTS RACLIST(SURROGAT) REFRESH
```

Figure 16-2 on page 373 shows the authentication flow when a Web client accesses IBM HTTP Server. After receiving an HTTP request from the browser, IBM HTTP Server verifies which security scheme is specified in the `UserID` directive of HTTP Server's configuration file (normally, `/etc/httpd.conf`).

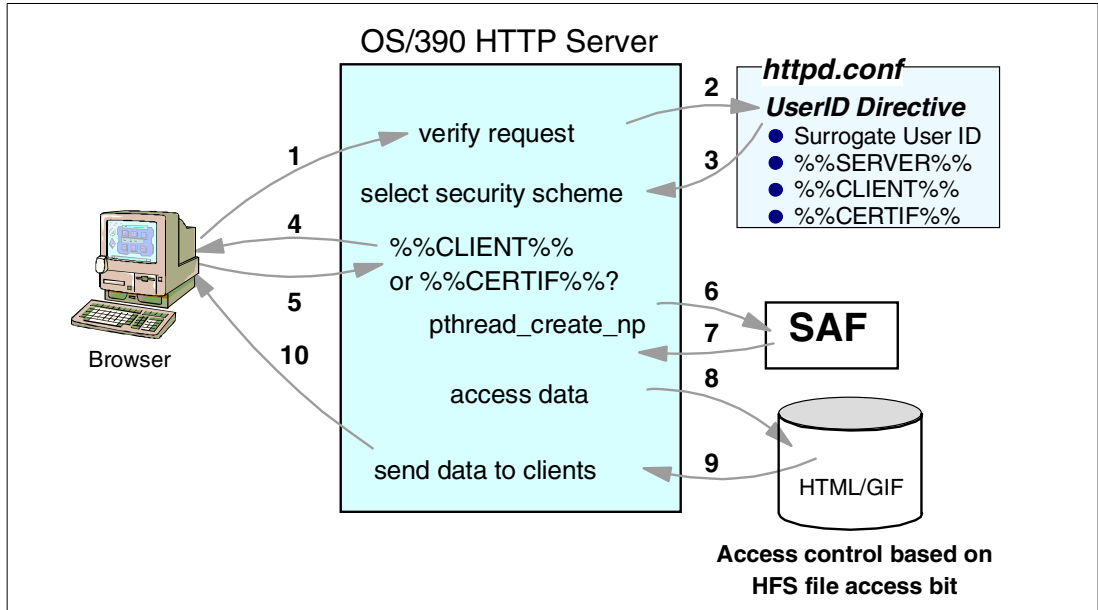


Figure 16-2 Authentication flow when accessing Web server

If a surrogate user ID is specified, IBM HTTP Server assigns the UID of the surrogate user ID to a thread for this request using the `pthread_security_np()` service. For example, if user ID EMPLOYEE with UID=100 is specified as a surrogate user ID, a worker thread runs with UID=100. If %%SERVER%% has been specified, z/OS HTTP Server assigns the HTTP Server's UID, that is UID=0 in most implementations, to a thread. In either case, z/OS does not require any more authentication information, such as a user ID/password or a client X.509 digital certificate.

Note: IBM HTTP Server can run under an authority other than superuser. See APAR PQ41777 for detailed information.

On the other hand, if %%CLIENT%% or %%CERTIF%% is specified, the client is required to submit security information. In the case of %%CLIENT%%, the Web browser displays a window such as that shown in Figure 16-3, which prompts the user to enter his or her user ID and password. Then IBM HTTP Server creates a worker thread with a UID of this RACF user ID.

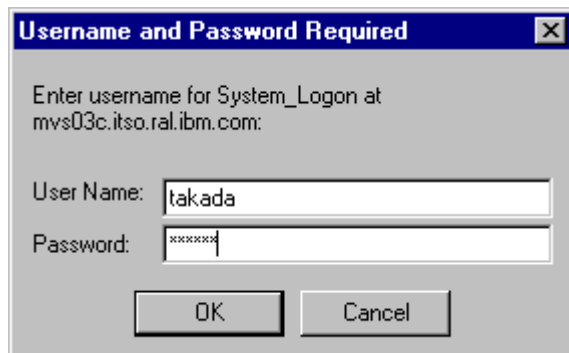


Figure 16-3 Netscape Navigator's pop-up window for the user ID and password

If %%CERTIF%% is specified, client authentication is required. A client must submit his or her X.509 client certificate and this certificate must be associated with his or her RACF user ID. After authentication is completed successfully, IBM HTTP Server creates a worker thread with the UID of this RACF user ID.

In the case of %%CLIENT%% or %%CERTIF%%, the worker thread's user ID is the RACF user ID that a user enters in a window or is associated with a client certificate, and accessing resources such as HTML files, GIF images, and CGI programs is controlled by HFS file attributes (unless already restricted via a DefProt or Protect statement).

The ability to have a different UID for each thread within a single process is considered superior to the schemes used by other UNIX-based Web servers. In other UNIX environments, Web servers cannot have threads with different UIDs within a process. Web servers must normally be assigned root authority in order to have sufficient privileges to open port 80 (the default HTTP protocol port). Thus, if worker threads are running in the same process as a Web server, all worker threads acquire root authority. Obviously, the fewer processes running under root authority, the more secure the system.

One solution to this is that used by the Apache Web server. It runs a main process under root, which in turn spawns child processes that change their user ID entity to whatever user and group are configured in the server's configuration file (see Figure 16-4). In this figure, these child processes run under the user "www".

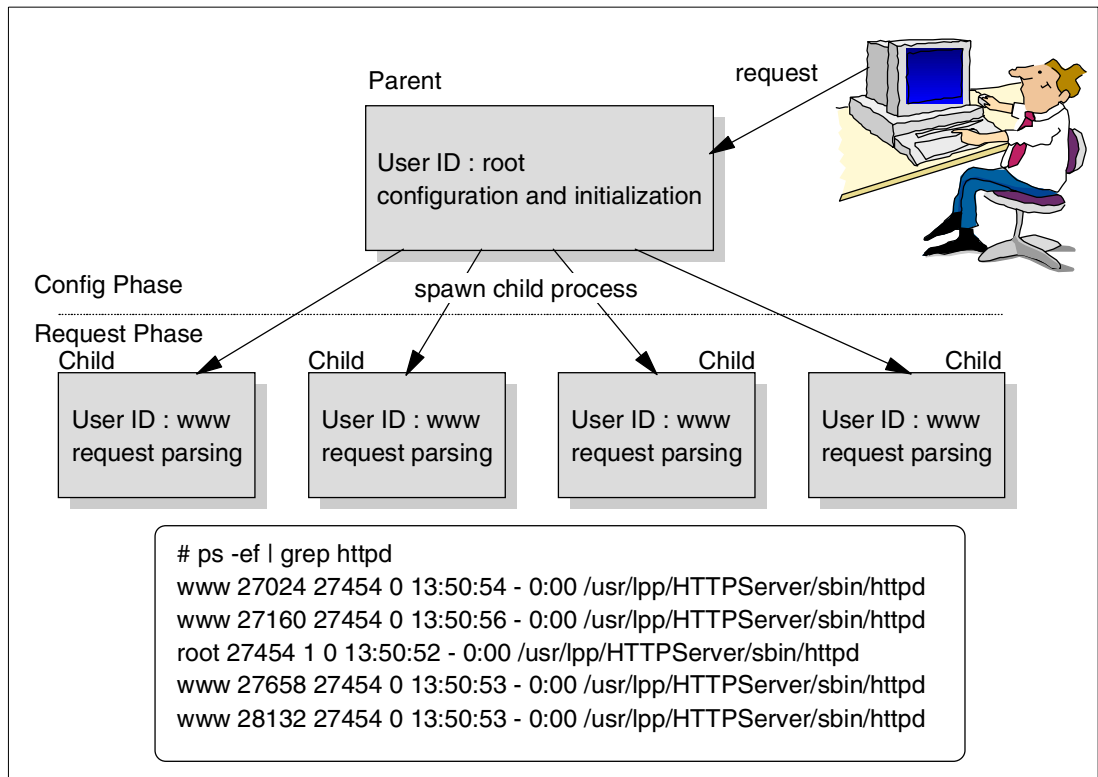


Figure 16-4 Alternative Web server process structure

In other UNIX platforms, Web administrators have to define access authority for requests from clients in advance. As a consequence every access authority has the same security information. On the other hand, in the UNIX System Services environment, a different access authority for each request can be created dynamically. In addition to the scalable server function, which allows IBM HTTP Server to run in different address spaces and define different performance goals for each workload type (simple HTML, CGI, Server API, Servlet and so forth), thread level security is one of the biggest advantages of the z/OS solution.

In critical Web environments such as online retailing, Internet banking and so on, the functions of changing access authority dynamically and workload management are indispensable. Because IBM HTTP Server has these abilities, it is the most appropriate Web server for a high-volume and secure Web environment. For detailed information about scalable Web server function, see *z/OS Workload Manager Implementation and Exploitation*, SG24-5326.

16.4.2 Protection directives

The use of Protection directives allows the Web server to control and limit access to certain resources. Protection directives can also be used to authenticate a user or to change the default user ID for the access of certain resources. In this section we will show you some examples of Protection directives and how they work.

Protection directives work based on the URL definition and can protect a specific file or resource, or protect everything beyond a certain directory. Because Protection directives work on the URL level and we have the ability to map real resources or files to the URL using Pass or Exec configuration directives, we are able to set up different Protection directives for the same resource or file. This can be achieved by accessing the real resource using two or more different URLs and mapping them accordingly.

Note: When using Fast Response Cache Accelerator (FRCA), all pages that are placed in cache are accessible to any browser - even pages that originally required authentication! Consequently, any documents that need to be protected and have a Protection or Protect directive that specifies a Map value of "All" or "Anybody" should be listed on a FRCANoCaching directive. Of course, documents served over an SSL connection are never cached. See APAR PQ40310 for more information.

When Protection directives require verification, they may use the same schemes and keywords as described in 16.4.1, "Access control directives" on page 371. They can also extend this possibility to groups and group files and certain parts of digital certificates, and can check passwords against RACF as well as against webmaster-maintained password files.

In the following examples, we show some typical access control definitions.

```

Protection IMW_Admin {
    ServerId      IMWEBSRV_Administration
    AuthType      Basic
    PasswdFile    %%SAF%%
    Mask          WEBADM,webadm,TAKADA,takada
}

Protect /admin-bin/* IMW_Admin WEBADM
Protect /reports/*  IMW_Admin WEBADM

Pass  /admin-bin/webexec/* /usr/lpp/internet/server_root/admin-bin/webexec/*
Exec  /admin-bin/*         /usr/lpp/internet/server_root/admin-bin/*
Pass  /reports/javelin/*  /usr/lpp/internet/server_root/pub/reports/javelin/*
Pass  /reports/java/*     /usr/lpp/internet/server_root/pub/reports/java/*
Pass  /reports/*         /usr/lpp/internet/server_root/pub/reports/*

```

Figure 16-5 RACF user ID/password authentication

The definition shown in Figure 16-5 is an example of how to protect access to URL `http://hostname/admin-bin/*` or `http://hostname/reports/*`. Pass and Exec directives map a URL to an HFS file path. The Protect directive defines which URLs should be authenticated and with whose user authority this request should be processed. In this example, when a client accesses the URL `/admin-bin/*` or `/reports/*`, the authentication specified in the Protection directive with label name "IMW_Admin" is done and this request is processed with WEBADM's UID. Because user ID WEBADM is a surrogate user ID, you must define RACF SURROGAT class profile BPX.SRV.WEBADM and give IBM HTTP Server's user ID READ authority to this profile.

The Protection directive defines the authentication method (AuthType and PasswdFile), realm (Server ID), and who is permitted to access the resources protected by this Protection directive (Mask). In this case, authentication is done by a SAF interface, that is, RACF. A client must input the user ID WEBADM or TAKADA and its valid password. Note that the user IDs are case sensitive. As a precaution, webadm and takada are also explicitly specified in the Mask subdirective.

Note also that the user IDs are used for the purposes of authentication only and are not related to the user ID assigned to the user request. As you can see in Figure 16-6, the Server ID specified in the Protection directive is displayed in the window to ask the client's user name and password.



Figure 16-6 Netscape Navigator's user authentication window

Once the authentication for this server ID is successful, when accessing a realm with the same server ID, you do not need to input a user ID and password again as long as you do not shut down the browser or clear the browser's memory cache.

Group files (GroupFile subdirective) are not compatible with RACF user IDs and group names. Further, password files (PasswdFile subdirective) are not advisable when RACF is available to handle the equivalent function. In addition, the maintenance of separate user lists is not advisable. As such, group and password files are not recommended.

What about LDAP with a password file? While this is a possibility, authentication would need to occur on each single resource request. As such, LDAP would need to be consulted every time you wanted to view a new URL. The performance impact would be so great that this option is not advisable.

In summary, use your local SAF product for user ID and password control.

The Protection, Protect and DefProt directives in the httpd.conf file have more subdirectives than we can show in this redbook. For more information about the Protection, Protect and DefProt directives in the httpd.conf file, see *z/OS V1R1.0-V1R3.0 IBM HTTP Server Planning, Installing, and Using V5.3*, SC34-4826.

16.5 Accessing back-end applications

If you want users to access back-end applications such as DB2, IMS, or CICS through the HTTP Server, you need to take into consideration the authority to access these resources.

Figure 16-7 shows how a user ID from a browser is passed to Net.Data through the IBM HTTP Server. This user ID is authenticated by IBM HTTP Server based on User ID, Protection or Protect directives. This user ID is used to check the authority to access DB2 resources. Therefore, to access DB2 resources via the HTTP Server, your user ID needs the authority to access not only the HTTP Server resources but also DB2 resources.

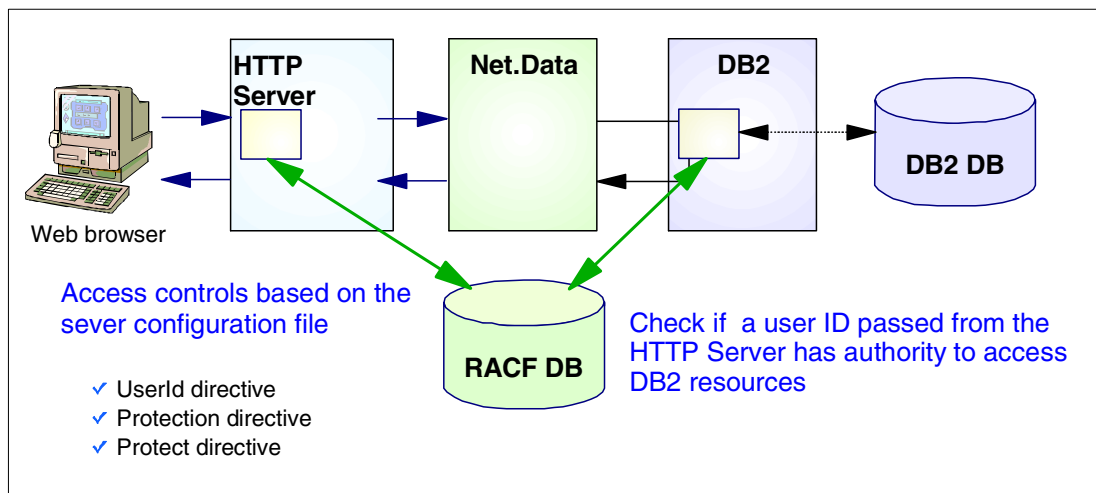


Figure 16-7 User ID passed to Net.Data from IBM HTTP Server

Figure 16-8 on page 378 is an example of the access control needed to query DB2 data through IBM HTTP Server and Net.Data.

If clients access the resource with a label DB2_Query on the Protect directive (1), a Protection directive with the same label (2) is used to decide how the access control for this resource should be done. In this case, you have to enter your RACF user ID and password because %%SAF%% is specified in the PasswdFile subdirective (3). The Mask subdirective specifies All (4) which means that all users defined in the password file (the RACF database) who are successfully authenticated will be allowed to execute the Net.Data connector. At that time, the user ID associated with the Net.Data request is the client's user ID because %%CLIENT%% has been specified in the UserID subdirective (5). Therefore, each client will need sufficient access to the DB2 objects needed to execute the query.

```

Protection DB2_Query {2
  ServerIdNetData_DB2_Access
  AuthTypeBasic
  UserID %%CLIENT%%5
  PasswdFile%%SAF%%3
  MaskAll4
}

Protect /netdata-cgi/db2www DB2_Query1

```

Figure 16-8 Controlling access to DB2 resources through the HTTP Server

For more information about Net.Data, see the following URL:

<http://www.ibm.com/software/data/net.data/>

16.6 SSL-related features in the IBM HTTP Server for z/OS

The SSL protocol was first implemented in the IBM Internet Connection Secure Server for z/OS (ICSS), introduced in September 1996. Since then, many enhancements have been made to the SSL support in Domino Go Webserver and IBM HTTP Server for z/OS, including SSL V3 support, support for the S/390 cryptographic hardware, and the possibility to associate RACF user IDs with client certificates used in SSL V3 client authentication.

16.6.1 Encryption support

Computer products containing encryption technology are subject to export and import controls by various national governments. Both Web servers and Web browsers are subject to these restrictions, depending on your geographic location.

For customers in most countries, U.S. Government export regulations are not a problem nowadays. Since October 2000, practically no restrictions for the export of cryptographic products into the EU countries and eight more countries exist. For most other countries, retail products can be exported to any customer. As a consequence, customers in all countries (except some restricted countries) should order the North America Security Feature.

The IBM HTTP Server for z/OS requires the specification of one of the following security features:

1. North America Security Feature (level 3):
 - Generate asymmetric keys from 512-1024 bits; encrypt data with symmetric keys from 40-168 bits; sign data with keys from 512-1024 bits; check signatures with keys from 512-1024 bits.
 - SSL V2 cipher specifications: Triple-DES; RC4 (128-bit); RC2 (128-bit); DES (56-bit); RC4 (40-bit); RC2 (40-bit).

- SSL V3 cipher specifications: Triple-DES SHA; RC4 SHA (128-bit); RC4 MD5 (128-bit); DES SHA (56-bit); RC4 MD5 (40-bit); RC2 MD5 (40-bit).
2. Export Security Feature (level 2):
- Generate asymmetric keys with a key size of 512 bits; encrypt data with symmetric keys from 40-56 bits; sign data with 512-bit keys; check signatures with keys from 512-1024 bits.
 - SSL V2 cipher specifications: DES (56-bit); RC4 (40-bit); RC2 (40-bit)
 - SSL V3 cipher specifications: DES SHA (56-bit); RC4 MD5 (40-bit); RC2 MD5 (40-bit).

The following table shows the feature codes for the IBM HTTP Server encryption features. Before OS/390 V2R10, three different encryption features were available; the third feature (France Security Feature, level 1) had a maximum symmetric key size of 40 bits.

Table 16-2 Cryptographic support FMIDs for IBM HTTP Server for z/OS

Product level	NA security	Export security	France security
HTTP Server 5.1 (HIMW510)	JIMW511	JIMW512	JIMW513
HTTP Server 5.2 (HIMW520)	JIMW521	JIMW522	JIMW523
HTTP Server 5.3 (HIMW530)	JIMW531	HIMW530	N/A

16.6.2 Global Server IDs

Before January 2000, U.S. Government export regulations did not allow the export of North America Security versions of Web browsers such as Netscape Communicator or Microsoft Internet Explorer to end users outside the United States and Canada. These Web browsers that would normally only use Export Security ciphers, could be enabled to use strong cryptographic algorithms in sessions with a Web server authorized to use strong encryption.

The U.S. Government had authorized VeriSign, Inc. and Thawte to issue special server certificates called *Global Server IDs* to those customers eligible to use strong encryption with servers outside the U.S. and Canada. These certificates are recognized by export versions of Microsoft Internet Explorer Version 3.02 and above, and by Netscape Navigator/Communicator Version 4 and above. When the Web browser recognizes the special certificate, it enables strong encryption routines such as RC4 with 128-bit keys or Triple DES with 168-bit keys. This process is sometimes also called *Server Gated Cryptography* (SGC).

The only reason Global Server IDs are still an issue worth considering is the fact that a large number of users outside the U.S. and Canada are still using export versions of the Web browsers, so a server with a Global Server Certificate can enable them to use strong encryption algorithms in communications with this server. Downloading a new Web browser can require more than 15 MB to be downloaded, which might frighten off users with only a modem connection.

16.6.3 Crypto hardware support for SSL

The Cryptographic Coprocessor Feature is available as a feature on the zSeries 900, the S/390 Enterprise Servers (9672), and S/390 Multiprise Servers.

To use this hardware feature, the z/OS Integrated Cryptographic Service Facility (ICSF) is needed. ICSF provides a cryptographic API that interfaces with the cryptographic hardware.

There is no specific setup for the SSL function to enable this support; once the server detects the presence of the crypto hardware feature, the encryption and decryption functions will be performed automatically using this feature.

The use of this hardware feature may help offload z/OS work for SSL. In an SSL handshake, the private key decryption process can take over 70 percent of the CPU cycles necessary to process the handshake. By using the cryptographic coprocessors, an S/390 G5 or G6 processor can service about 150 full SSL handshakes per second while still leaving most of the CPU power to the applications. If this is insufficient, PCI Cryptographic Coprocessor cards (PCICC) can be added to 9672 G5 and G6 systems and 2064 systems, which can increase the ability of the system to handle SSL handshakes considerably. A 9672 system equipped with eight PCICC cards can handle about 1000 full SSL handshakes per second, while a 2064 system equipped with eight PCICC features (16 cards) can handle about 2000 full SSL handshakes per second.

16.7 SSL scenario

There are two options available when establishing an SSL connection. They are server authentication (SSL V2 and V3) and client authentication (SSL V3). While server authentication is mandatory, client authentication is optional. In this section, we discuss how to establish server authentication between IBM HTTP Server and Netscape Communicator 4.6.

16.7.1 Server authentication

The steps to start your IBM HTTP Server as a secure server are:

1. Establish a key database or RACF keyring with a server certificate

The IBM HTTP Server can use a key database established with the gskkyman utility or it can use a RACF keyring. If a key database (a file in the HFS) is used, the password to the key database must be specified in a stash file to allow the server to use the key database. The password is scrambled but not encrypted which can be a security exposure because any superuser can read the stash file. A RACF keyring does not require a password, it can only be accessed by the server's user ID or a SPECIAL user. For examples on creating key databases or keyrings, see Appendix 10, "Certificate management in z/OS" on page 203.

2. Customize SSL-related directives in the HTTP Server configuration file

To make z/OS HTTP Server enable server authentication, you have to specify at least three directives in the httpd.conf file:

```
sslmode    on 1
sslport    443 2

keyfile    /u/takada/sslkey/server.kdb
keyfile    WEBRING SAF 3
```

Figure 16-9 The SSL-related directives in the HTTP configuration file

1 Use this directive to turn on SSL support.

2 Use this directive to set the port for SSL security (HTTPS). The default is 443.

3 Use this directive to specify the key database file. If you do not specify the name of your key database, IBM HTTP Server cannot bind to a secure port. Alternatively, you can specify the name of a RACF keyring.

3. Restart the HTTP Server

After the SSL-related directives in `/etc/httpd.conf` file shown in Figure 16-9 on page 380 are modified, you stop and restart the HTTP Server. The following result of the `onetstat -c` command shows the HTTP Server listening on a normal HTTP port 80 and a secure HTTPS port 443.

```
TAKADA @ RA03:/u/takada>onetstat -p TCPIP -c
MVS TCP/IP onetstat CS V2R8      TCPIP Name: TCPIP      15:39:15
User Id Conn      Local Socket      Foreign Socket      State
----- ----      -
ICAPCFG 000000F8 0.0.0.0..1014    0.0.0.0..0         Listen
INETD1  00000020 0.0.0.0..109     0.0.0.0..0         Listen
INETD1  0000001E 0.0.0.0..2023    0.0.0.0..0         Listen
INETD1  0000012B 9.24.104.33..2023 9.24.106.155..1100 Establish
INETD1  0000001F 0.0.0.0..110     0.0.0.0..0         Listen
TCPIP   00000011 127.0.0.1..1026  127.0.0.1..1025    Establish
TCPIP   00000016 0.0.0.0..8823    0.0.0.0..0         Listen
TCPIP   00000012 127.0.0.1..1025  127.0.0.1..1026    Establish
TCPIP   00000015 0.0.0.0..7723    0.0.0.0..0         Listen
TCPIP   00000014 0.0.0.0..6623    0.0.0.0..0         Listen
TCPIP   00000017 0.0.0.0..9923    0.0.0.0..0         Listen
TCPIP   0000000C 0.0.0.0..1025    0.0.0.0..0         Listen
TCPIP   00000013 0.0.0.0..23      0.0.0.0..0         Listen
WEBSRV  00000178 0.0.0.0..443     0.0.0.0..0         Listen
WEBSRV  00000177 0.0.0.0..80      0.0.0.0..0         Listen
ICAPIKE 000000FA 9.24.104.33..500 *..*                UDP
ICAPSLG 000000F7 0.0.0.0..514     *..*                UDP.
```

Now you can establish the SSL connection between IBM HTTP Server and a browser. You use the special URL format `https://hostname:port/` to request an SSL connection to an HTTP Server. If you do not specify a port number, the browser requests a connection to port 443 as a default.

Note: While it is possible to use other ports than 443 for SSL, this is likely to cause problems if clients will be accessing the server through firewalls with HTTP proxy servers. The reason is that a proxy server must be enabled for SSL tunneling by port number. Most proxies are only enabled for port 443 so they will block all SSL traffic to other ports without returning any error messages.

In Figure 16-10 on page 382, we show an example where Netscape establishes an SSL connection with our secure server.

To view the information in the server certificate, click **Security** on the menu button of the Netscape Navigator window. After the security information window is displayed, click **View Certificate**. Then the window that shows the server certificate information appears (see Figure 16-11 on page 382).



Figure 16-10 Example of the Netscape security screen with an SSL connection

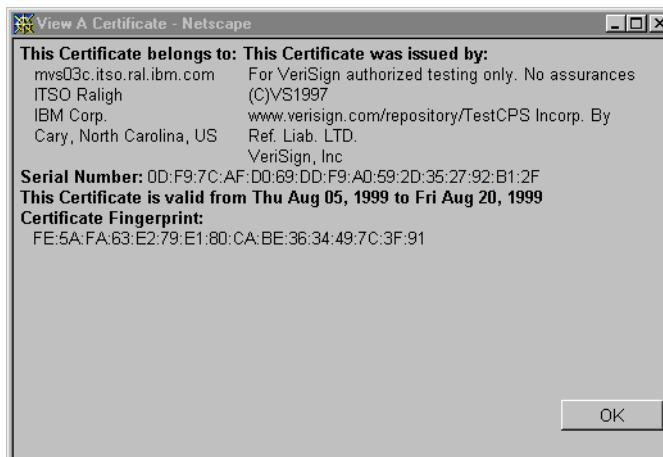


Figure 16-11 Server certificate information

16.7.2 Client authentication

SSL Version 3 is required for client authentication. If you set up your server for SSL client authentication, the server requests a certificate from all clients making an https request.

We obtained the VeriSign trial client certificate from VeriSign's Web site and used that as the client certificate for our client authentication SSL test. In the following discussion, we show how to establish an SSL V3 connection for this purpose.

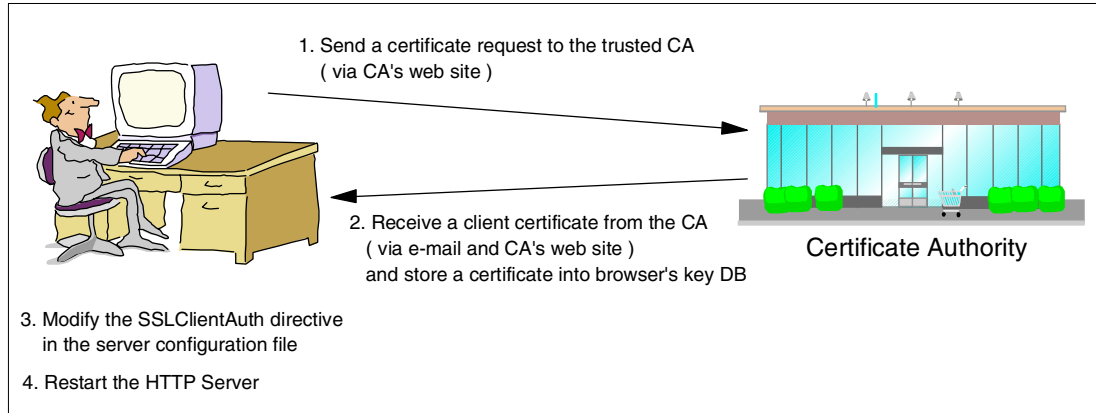


Figure 16-12 Flow to implement SSL client authentication

Figure 16-12 shows the steps that are usually needed to request a client certificate from a Certificate Authority (CA) and store the certificate in the browser's key database. The certificate can be sent by e-mail or downloaded from the CA Web site. To enable the use of client certificates in the Web server, the SSLClientAuth directive needs to be set to "local" and the Web server needs to be restarted.

To view your client certificate information, click the **Security** button on the Netscape Navigator's primary window (Figure 16-13), click **Yours** in the Certificates category, select the certificate you desire from the These are your certificates: scroll box, and click **View**. Then the window that shows your certificate information appears (see Figure 16-14 on page 384).

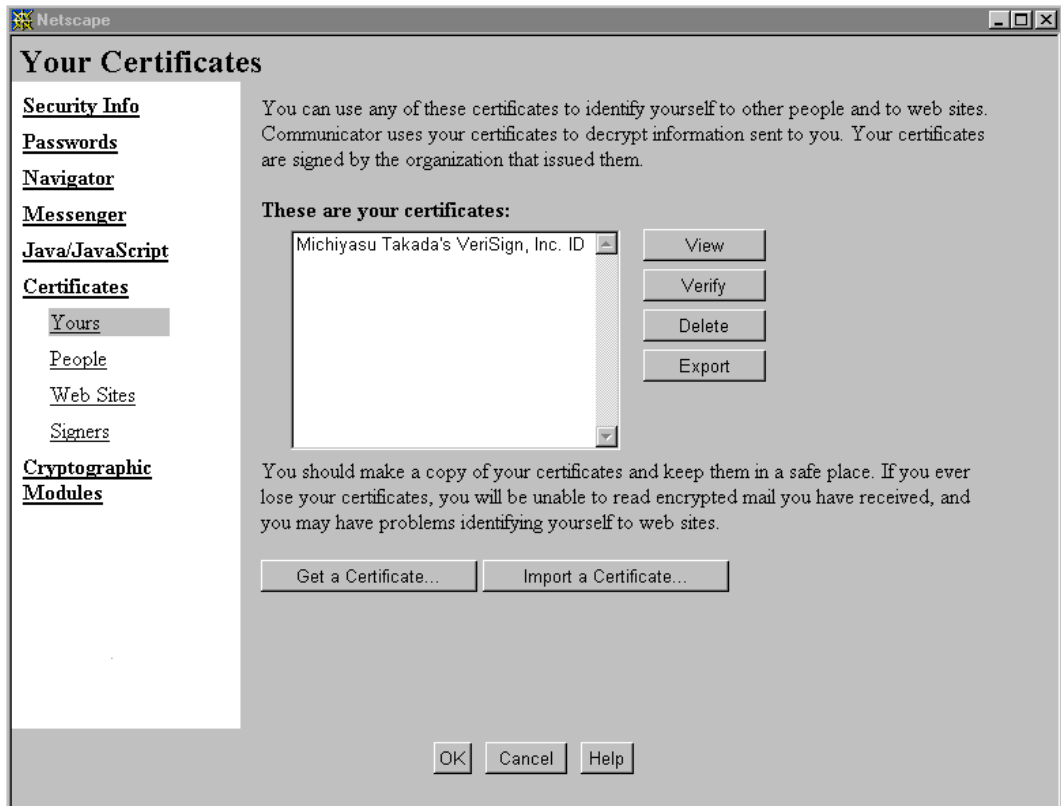


Figure 16-13 Security information window

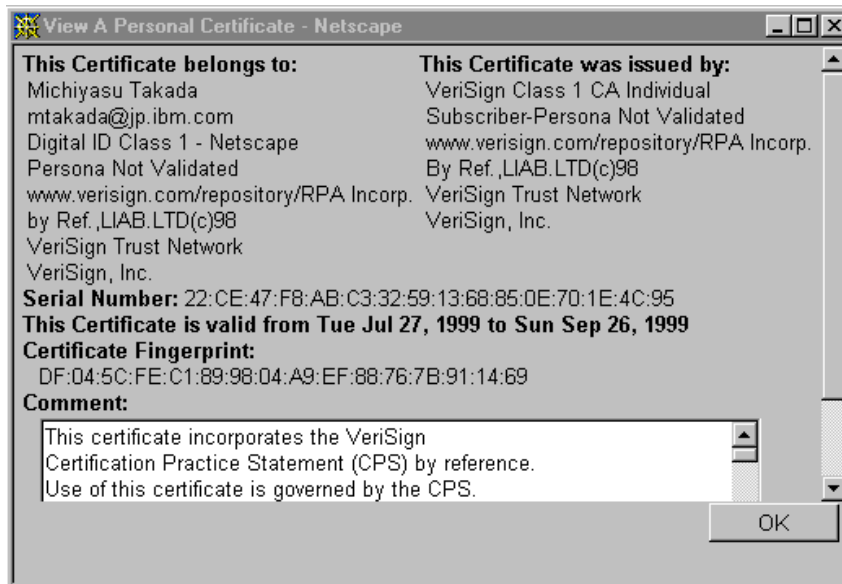


Figure 16-14 Client certificate information

With client authentication, the server can control access to resources by using the client certificate information instead of challenging for a user ID/password. This function is enabled by coding SSL authentication parameters in the right places.

First of all, you have to set the SSLClientAuth directive to on (the default is off). The SSLClientAuth directive has the following three options for client authentication:

- ▶ local
 The z/OS HTTP Server uses the trusted root CA certificate in the key database file stored locally for client authentication.
- ▶ strong
 The server uses the X.500 directory server specified in the SSLX500Host directive for client authentication.
- ▶ passthru
 When this option is specified, the server performs no validation of the client's certificate. In this case, alternative validation methods, such as a user -applied GWAPI or CGI routine, need to be available.

In our test environment, we changed this option to local. That is to say, IBM HTTP Server used a locally stored key database file to check if a root CA that signed a client certificate is really trusted.

```
SSLClientAuth local
```

The z/OS HTTP Server provides 13 possible SSL-specific Protection subdirectives and you can control the access of your sensitive resources based on any combination of these subdirectives.

The following six subdirectives are used to identify a client:

CommonName, Country, Locality, StateOrProvince, Organization, OrgUnit

The following six subdirectives are used to identify an issuer, that is, a CA:

IssuerCommonName, IssuerCountry, IssuerLocality, IssuerStateOrProvince, IssuerOrganization, IssuerOrgUnit

When the last and 13th one, `SSL_ClientAuth`, is used, the server permits access only if a client submits its certificate.

To confirm which identity information the client certificate gives to the z/OS HTTP Server, you should obtain the information using the `-vv` trace option provided by IBM HTTP Server:

```
Client certificate data:
Organization = VeriSign, Inc.
Organizational Unit = Digital ID Class 1 - Netscape
Issuer Common Name = VeriSign Class 1 CA Individual Subscriber-Persona Not Validated
Issuer Organization = VeriSign, Inc.
Issuer Organizational Unit = www.verisign.com/repository/RPA Incorpor. By Ref.,LIAB.LTD(c)98
```

Figure 16-15 Client certificate data obtained by `-vv` trace

You can define some SSL-specific Protection subdirectives based on the result shown in Figure 16-15.

Figure 16-16 shows an example of the protection based on the client certificate information. The use of these subdirectives does not create any association of the certificate owner with a RACF user ID. Therefore, a `Userid` subdirective or a user ID positional parameter on the `Protect` directive is needed to specify the surrogate user ID for this request. In our example, user ID `WEBADM` will be used.

```
Protection IMW_Admin {
    ServerId      IMWEBSRV_Administration
    AuthType      Basic
    Organization   "VeriSign, Inc." 1
    OrgUnit       "Digital ID Class 1 - Netscape"
    IssuerOrganization "VeriSign, Inc."
    IssuerOrgUnit  "www.verisign.com/repository/RPA Incorpor. By Ref.,LIAB.LTD(c)98"
    Mask          anybody@(*) 2
}

Protect /admin-bin/* IMW_Admin WEBADM
Protect /reports/*   IMW_Admin WEBADM

Pass  /admin-bin/webexec/* /usr/lpp/internet/server_root/admin-bin/webexec/*
Exec  /admin-bin/*         /usr/lpp/internet/server_root/admin-bin/*
Pass  /reports/javelin/*  /usr/lpp/internet/server_root/pub/reports/javelin/*
Pass  /reports/java/*     /usr/lpp/internet/server_root/pub/reports/java/*
Pass  /reports/*         /usr/lpp/internet/server_root/pub/reports/*
```

Figure 16-16 Access control using SSL V3 client authentication

1 Specify one or more certificate identities. We selected `Organization`, `OrgUnit`, `IssuerOrganization`, and `IssuerOrgUnit` for client authentication. Unless the certificate identities the client submits completely match this information, the authentication fails. If you have specified the `-vv` trace option in the server's start procedure, you will see the following message:

```
Forbidden... SSL Client Authentication failed for URL /usr/lpp/internet/server
_root/admin-bin/webexec/cfginit.html for 9.24.106.145.
The Client Certificate subject DN is not valid
```

- 2 Only if the client certificate data matches the entry specified in the SSL-related Protection subdirectives does the server permit the client to access this resource. You can control this more strictly by defining a specific user common name.

When you access an IBM HTTP Server that has client authentication enabled, you receive the window that prompts you to select a certificate (see Figure 16-17).

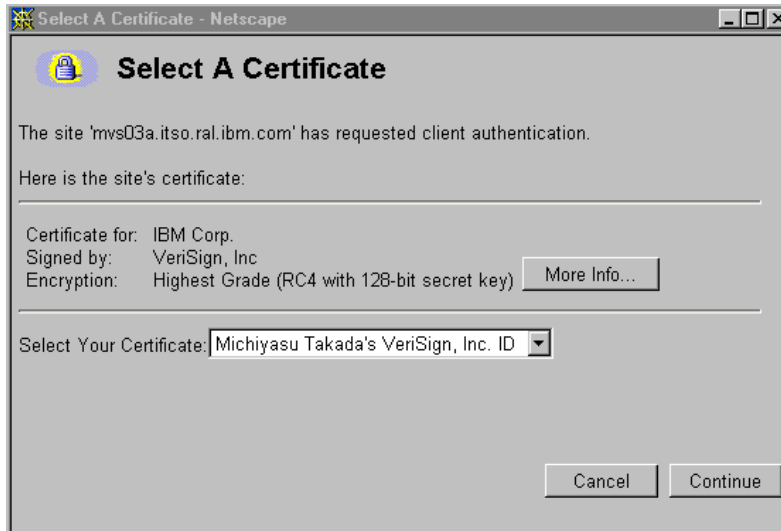


Figure 16-17 Netscape Navigator's window requiring your certificate

In Figure 16-17, after you click **Continue**, you will see the pop-up window shown in Figure 16-18 if you set a password for your certificate. We strongly recommend you set a password when storing your certificate into your browser. If you do not, someone else could easily use your certificate while you are away from your workstation. Here, you must enter a valid password. This password is what you entered when you first stored your new certificate.

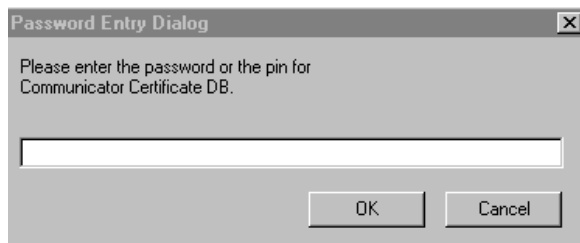


Figure 16-18 Enter a password for your certificate

After you succeed in client authentication, you can establish an SSL connection and communicate with IBM HTTP Server for z/OS securely.

IBM HTTP Server for z/OS provides the PKIServ utility, which is a Web-based utility that enables an installation to issue client certificates to users with a user ID in RACF. For more information, see the documentation for APARs OW45211/OW45212 and the following URL:

<http://www-1.ibm.com/servers/eserver/zseries/zos/racf/webca.html>

16.8 Associating a client certificate with a RACF user ID

The IBM HTTP Server is able to execute Web requests with a user ID retrieved from RACF on the basis of the client certificate received during an SSL handshake. The following example shows how the parameter `%%CERTIF%%` is used in a Protection directive for this purpose:

```
Protection very_secure {
    ServerId      CICS_Project
    AuthType     Basic
    SSL_ClientAuth client
    UserId       %%CERTIF%%
    PasswdFile   %%SAF%%
    Mask        anybody
}
```

The Mask value of “anybody” is needed to prevent the Web server from prompting the client for user ID and password. If “All” is used, the user will have to enter a valid user ID and password in addition to the client certificate.

The SSL_ClientAuth value of “client” will cause all requests from clients who cannot present an acceptable certificate to fail with a “403 Forbidden” status code. If this subdirective has not been specified, the Web server will fall back to `%%CLIENT%%` processing if the client does not provide a valid certificate. The client certificate must be signed by a CA whose root CA certificate is contained in the Web server’s key database or RACF keyring to be acceptable.

16.8.1 RACF digital certificate support

The security server component that was first shipped in OS/390 Version 2 Release 4 introduced the ability to verify digital certificates and to assign a user ID based on the certificate. The digital certificate and the user-related information are stored in the RACF class DIGTCERT. The RACDCERT command can be used to add, alter, list, or delete DIGTCERT profiles.

The digital certification flow is:

- ▶ IBM HTTP Server passes the client’s digital certificate to UNIX System Services. It is assumed that the Web server verifies the validity of the client certificate. This means that the server has verified that the certificate is genuine and was issued by a trusted authority, that its validity date is current, and that the client is the owner of the certificate.
- ▶ UNIX System Services passes the certificate as a parameter into RACF’s initACEE callable service.
- ▶ RACF extracts information from the digital certificate, identifies a RACF user ID from it, and builds the ACEE.

For detailed information, see *z/OS V1R2 Security Server (RACF) Security Administrator’s Guide*.

16.8.2 Install and maintain digital certificates in RACF

To install a digital certificate in RACF you need to enable the RACF DIGTCERT class and add it to the RACLIST:

```
SETR CLASSACT(DIGTCERT) RACLIST(DIGTCERT)
```

To maintain DIGTCERT, the command RACDCERT is used and needs to be defined in the AUTHCMD NAMES list in the IKJTSOxx member of SYS1.PARMLIB. The RACDCERT command is a RACF TSO command used to:

- ▶ List information about the existing certificates for a specific RACF-defined user ID, or your own user ID.
- ▶ Add a certificate definition and associate it with a specified RACF-defined user ID, or your own user ID, and set the TRUST flag.
- ▶ Alter the TRUST flag for an existing definition.
- ▶ Delete an existing definition.

To facilitate the addition of a certificate definition, the input to RACDCERT for the ADD function is the name of a data set that contains the digital certificate.

To issue the RACDCERT command, you must have one of the following RACF authorities or permissions:

- ▶ The SPECIAL attribute
- ▶ Sufficient authority to RACF resource IRR.DIGTCERT.function in the FACILITY class, where function is LIST, ADD, ALTER or DELETE
 - READ access to IRR.DIGTCERT.function to perform the function for yourself
 - UPDATE access to IRR.DIGTCERT.function to perform the function for others

The following setting shown in Figure 16-19 allows a user to register a certificate with his or her own RACF user ID:

```
RDEFINE FACILITY IRR.DIGTCERT UACC(NONE)
PERMIT IRR.DIGTCERT.LIST CLASS(FACILITY) ID(TAKADA) ACCESS(READ)
PERMIT IRR.DIGTCERT.ADD CLASS(FACILITY) ID(TAKADA) ACCESS(READ)
PERMIT IRR.DIGTCERT.ALTER CLASS(FACILITY) ID(TAKADA) ACCESS(READ)
PERMIT IRR.DIGTCERT.DELETE CLASS(FACILITY) ID(TAKADA) ACCESS(READ)
SETROPTS RACLIST(FACILITY) REFRESH
```

Figure 16-19 Protect RACDCERT command

Whenever you issue a RACDCERT ADD, ALTER or DELETE, you should refresh the DIGTCERT class by issuing the SETROPTS RACLIST(DIGTCERT) REFRESH command. If you do not refresh the RACLISTed DIGTCERT profiles, RACF will still use the new digital certificate. However, performance may be affected.

The TRUST and NOTRUST parameters are used to specify whether the mapping of a certificate to a RACF user ID is valid or not.

There are two ways to register a digital certificate with RACF:

- ▶ Using the TSO command interface with the RACF command RACDCERT.
- ▶ Using the RACF self-registration Web application distributed in OS/390 V2.6 and later in SYS1.SAMPLIB (members RACINSTL and IRR31933).

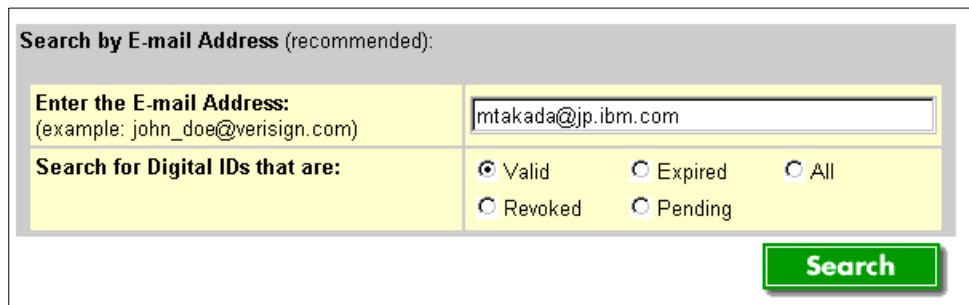
16.8.3 Register a certificate using RACDCERT

Unfortunately there is more than one format in which a digital certificate can be stored, and not all versions of all Web servers and browsers support the same selection of formats. A prime example is that OS/390 V2R7 and earlier releases do not support the format known as Public Key Cryptography Standards (PKCS) #12, whereas the popular browsers use precisely this format. Therefore, you may need to obtain your client certificate in the PKCS#7 format supported by older releases of OS/390. OS/390 V2R8 and later releases support PKCS#12.

You can obtain a copy of your certificate in any format from the Certificate Authority. Usually this is done using a browser. To download a file in PKCS#7 format you have to go to your Certificate Authority home page, do a search for your valid certificates, and download your certificate again with this option.

We show how to obtain a PKCS#7 format certificate from the VeriSign Web site:

<https://digitalid.verisign.com/services/client/index.html>



The screenshot shows a search form titled "Search by E-mail Address (recommended):". It contains a text input field with the email address "mtakada@jp.ibm.com" entered. Below the input field, there are radio buttons for "Valid", "Expired", "All", "Revoked", and "Pending". The "Valid" radio button is selected. A green "Search" button is located at the bottom right of the form.

Figure 16-20 Search for digital ID

You can search for your certificate by e-mail address, name or Digital ID Serial Number and Issuer Name at the Web page shown in Figure 16-20. Then the Web page that contains certificate information that met the criteria appears. From this page, you can download various certificate formats including PKCS #7 format (Figure 16-21 on page 390).

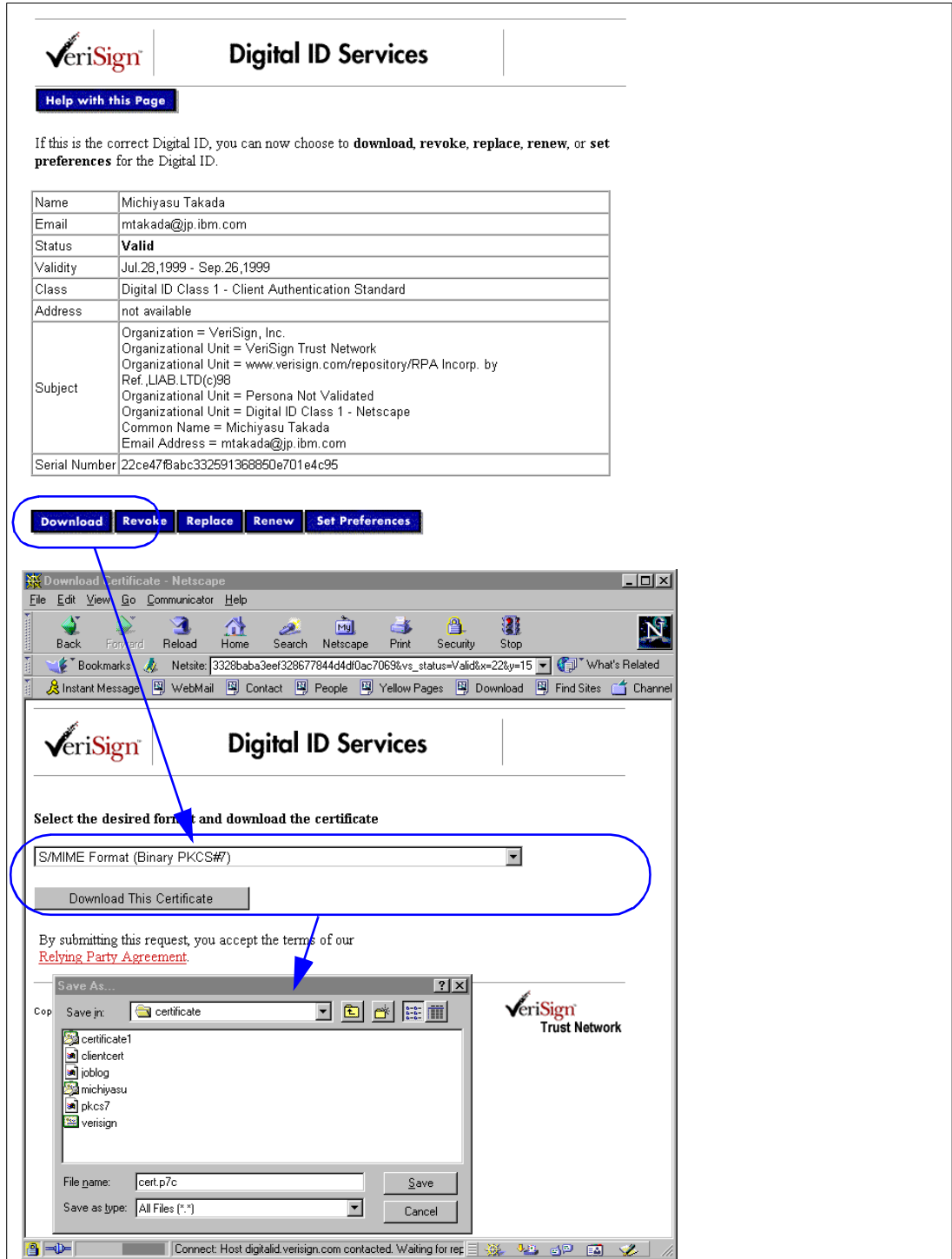


Figure 16-21 Download a certificate in PKCS#7 format from VeriSign Web site

If your OS/390 system is V2R8 or later, you can use PKCS#12 format to register your certificate into the RACF database. In this case, we show (Figure 16-22 on page 391) how you can simply extract your certificate in PKCS#12 format from the Netscape browser.

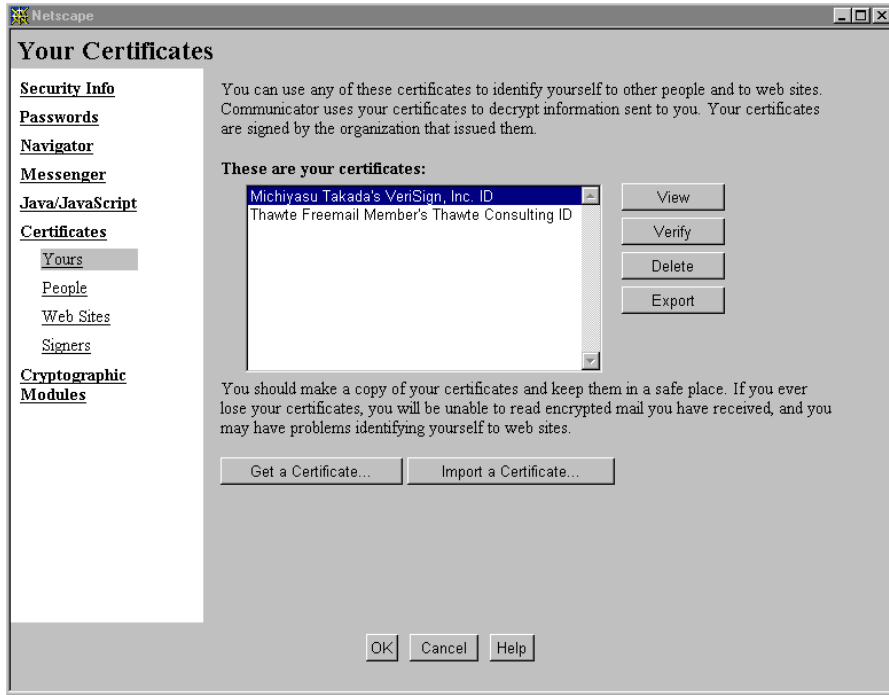


Figure 16-22 Exporting your client certificate in PKCS#12 format

First click the **Security** button on the Navigation Toolbar. After the Your Certificates window shown in Figure 16-22 appears, click **Yours** under Certificates. Then, under These are your certificates, select the client certificate you desire and click **Export**. Now you are required to enter a password for your browser's certificate database, which you originally specified in Figure 16-18 on page 386.

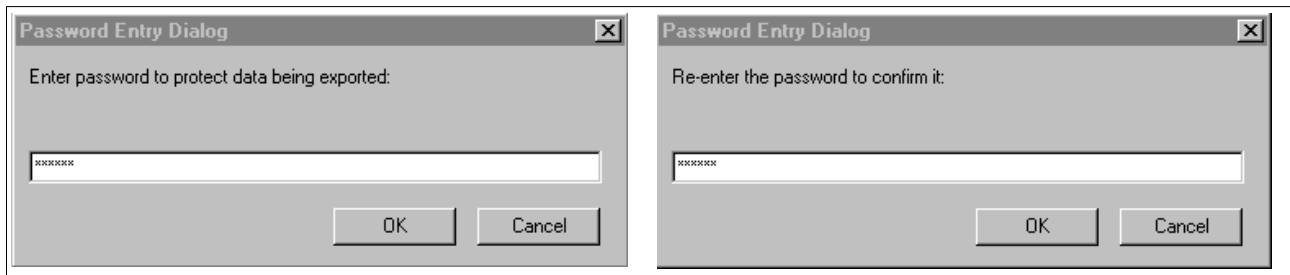


Figure 16-23 Setting a password for your certificate

Then you are required to enter a password for your certificate. This password is required later to issue the RACF command RACDCERT ADD.

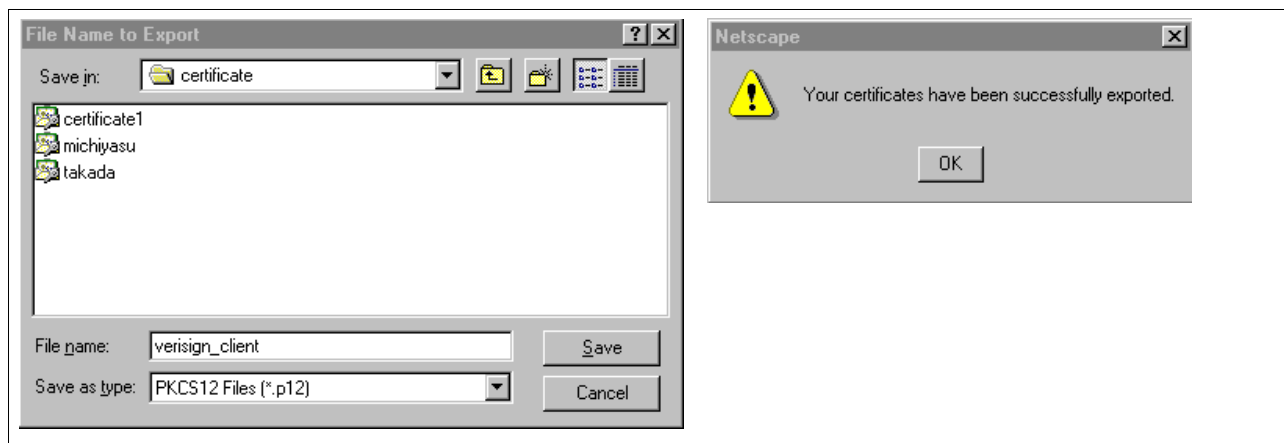


Figure 16-24 Storing your certificate in your workstation

Finally store your exported certificate in a local file. You will need to transfer this file to an MVS data set as a binary file, using or NFS or some other means.

To use the RACDCERT command the file must be in an MVS sequential data set. We recommend you allocate a five-to-ten track sequential data set using the following DCB attributes:

```
DSORG=PS
RECFM=VB
LRECL=4096
BLKSIZE=27998
```

Register the certificate using the RACDCERT TSO command:

```
raccert id(takada) add('takada.certif.pkcs12') trust password('password')
```

Figure 16-25 Register a certificate in the RACF database

This example registers the certificate stored in the sequential data set TAKADA.CERTIF.PKCS12 with the user ID TAKADA. If you registered a certificate in PKCS #12 format and forgot to specify the password option, you will see the following message:

```
IRRD127I The data set contains a PKCS12 encrypted certificate. The PASSWORD keyword must be specified to process the certificate. The certificate is not processed.
```

This password is the one specified in Figure 16-23 on page 391. If you are using PKCS #7 format, you do not have to specify any password for your certificate.

You will see the following message when the request has been completed successfully:

```
IRRD119I Certificate Authority not defined to RACF. Certificate added with TRUST status.
```

The message is saying that you are adding a certificate to RACF and RACF does not know about the certificate.

You may now list your certificate issuing the RACDCERT LIST command as shown in Figure 16-26 on page 393.


```

RACDCERT ID(TAKADA) LIST

Digital certificate information for user TAKADA:

Label: Michiyasu Takada's VeriSign, Inc
Status: TRUST
Start Date: 1999/07/27 20:00:00
End Date: 1999/09/26 19:59:59
Serial Number:
>22CE47F8ABC332591368850E701E4C95<
Issuer's Name:
>CN=VeriSign Class 1 CA Individual Subscriber-Persona Not Validated.OU<
>=www.verisign.com/repository/RPA Incomp. By Ref.,LIAB.LTD(c)98.OU=Ver<
>iSign Trust Network.O=VeriSign, Inc.<
Subject's Name:
>mtakada@jp.ibm.com.CN=Michiyasu Takada.OU=Digital ID Class 1 - Netsca<
>pe.OU=Persona Not Validated.OU=www.verisign.com/repository/RPA Incomp<
>. by Ref.,LIAB.LTD(c)98.OU=VeriSign Trust Network.O=VeriSign, Inc.<
Private Key Type: Non-ICSF
Private Key Size: 512
Ring Associations:
*** No rings associated ***

```

Figure 16-26 Displaying certificate information associated with RACF user ID

16.8.4 Certificate self-registration with RACF

You can register your certificate in the RACF database without issuing the RACDCERT command. You may use the application to enable autoregistration of digital certificates, RACF Self-Registration (RAR). A sample window from the application is shown in Figure 16-27 on page 394.

Storing client certificates in the RACF database is an excellent way to create a one-to-one mapping between the client certificates of employees and their user IDs, but it does not scale well in the case where, for instance, large numbers of customer certificates need to be mapped to a small number of surrogate user IDs.

16.8.5 Certificate name filtering

z/OS provides support for associating RACF user IDs with client certificates, called *certificate name filtering*.

Using certificate name filtering, client certificates are not stored in the RACF database. The association between one or more certificates and a RACF user ID is achieved by defining a filter rule that specifies parts of the distinguished name of the certificate owner and/or issuer (CA) that need to be matched for the user ID to be assigned. A sample filter rule might look like the following:

```
RACDCERT ID(DEPT3USR) MAP SDNFILTER('OU=DEPT3.OU=NY.OU=Sales.O=XYZ Co')
```

This sample filter rule would associate user ID DEPT3USR with all certificates when the distinguished name of the owner has the organization "XYZ Co" and the organizational units "DEPT3", "NY", and "Sales" in it.

For such applications as the z/OS Web server, the TN3270 server, or the FTP server in z/OS V1R2, which can call the RACF interface to get a user ID that is associated with a client certificate, it is transparent whether the certificate was stored in the RACF database or a certificate name filter rule was used.

Response



Verify your Certificate Information

Your RACF UserID:	BOCHE
Certificate Common Name:	Ulrich K Boche
Country:	
State or Province:	
Locality:	
Organization:	VeriSign, Inc.
Organizational Unit:	Digital ID Class 1 - Netscape Full Service
Issuer Common Name:	VeriSign Class 1 CA Individual Subscriber-Persona Not Validated
Issuer Country:	
Issuer State or Province:	
Issuer Locality:	
Issuer Organization:	VeriSign, Inc.
Issuer Organizational Unit:	www.verisign.com/repository/RPA Incorp. By Ref.,LIAB.LTD(c)98
<input type="button" value="Register Personal Certificate"/> <input type="button" value="Deregister Personal Certificate"/>	

If the data is not valid, click here to go back to the previous page [CANCEL](#)

The data and information presented are for demonstration purposes only. © Copyright IBM Corporation 1998

Figure 16-27 Certificate self-registration application sample window

16.9 Retrieving LDAP information

As discussed in “Authentication with the z/OS LDAP server” on page 402, centralized user management is one of the key roles LDAP was designed to provide. That includes support for the synchronization of user IDs and passwords. In Lotus Domino Go Webserver for OS/390 Release 5.0, LDAP support, that is to say the ability to access the LDAP server for user authentication, was introduced. This means that if authentication information (such as a set of user IDs and passwords or user certificates) has been stored in the LDAP directory, users can use this information to access protected resources in IBM HTTP Server. This authentication information can be used not only by IBM HTTP Server for z/OS but also by any other LDAP-enabled applications for the purposes of user authentication. In addition, using LDAP support allows multiple IBM HTTP Servers for z/OS to share configuration information.

IBM HTTP Server LDAP support is extremely scalable. You have a choice of LDAP configurations including:

- ▶ A single LDAP server.

- ▶ High availability configurations using multiple LDAP servers, for example, primary, secondary, and so on.
- ▶ Different LDAP servers to be accessed for different requests. For example, requests may come from two different IP addresses, and the Web server could contact a different LDAP server for each one.

You do not necessarily need to select z/OS LDAP Server as your repository for IBM HTTP Server LDAP support. Any other LDAP servers can interact with IBM HTTP Server. However, the z/OS LDAP Server provides various scalable features, such as exploiting DB2 and/or RACF back-end server, Parallel Sysplex support, connection optimization and so forth. Therefore, in an environment where many different servers exploit LDAP for authentication, it is better for you to implement z/OS LDAP Server for centralized user management.

In the following example, we show you how to extract user IDs and passwords stored in the LDAP server.

16.9.1 Configuring LDAP on IBM HTTP Server

To provide the server with information about the LDAP servers in which to store information, you have to use the LDAPInfo directive. Storing information on an external LDAP server allows applications to share the same information.

```
LDAPInfo ITSOLDAPSRV 1 {
  Host 9.24.104.113 2
  Transport TCP 3
  ClientAuthType Basic 4
  ServerAuthType Basic 5
  ServerDN "cn=HTTP Server,ou=ITSO,ou=Raleigh,o=IBM_US,c=US" 6
  ServerPasswordStashFile /u/takada/ldap/ldappasswd 7
  UserNameFilter "(&(objectclass=organizationalPerson)(cn=%v1* %v2*))" 8
  UserSearchBase "o=IBM_US, c=US" 9
}
```

Figure 16-28 Specifying the LDAP server information

1 This is the name you want to associate with this LDAP server setup. The name can then be used by subsequent Protection directives to point to this LDAP setup.

2 Specify the host name of the LDAP server you want to exploit for authentication.

3 Specify the port number on which the LDAP server listens. The default port is 389 for non-SSL-transported connections, and 636 for SSL-transported connections.

4 Specify the authentication type for connections made on behalf of the client. Basic means the client must provide a user ID and password in order to authenticate. Another option is “Cert”, meaning that the client’s certificate is used for authentication.

5 Specify the authentication type for the server’s connection to the LDAP server. “Basic” means that if IBM HTTP Server logs in to the LDAP server, it uses its distinguished name from the ServerDN subdirective, and the password provided in the password stash file. If you specify “None”, the LDAP server will allow anonymous access.

6 This is the distinguished name of the HTTP Server. This name is used when accessing the LDAP server when ServerAuthType is set to “Basic”. You have to store this distinguished name in the LDAP server.

7 This is the file containing the encrypted password to access the LDAP server. The password is only used if ServerAuthType is Basic. Create the stash file using the **htadm** command or using the Configuration and Administration forms. The syntax of the **htadm** command to create a stash file is:

```
htadm -stash file_name password
```

This password must be registered as an attribute (normally userPassword attribute) in the LDAP server.

8 Specify the filter used to convert the user name as input by the user to a search filter for an LDAP entry. The default is "(&(objectclass=person)(cn=%v1* %v2*))" where %v1 and %v2 are the words typed by the user. For example, if the user types "Cameron Diaz" in the window shown in Figure 16-3 on page 373, the resulting search filter would be "(cn=Cameron* Diaz*)".

9 Specify the starting point for the LDAP server to search for user names.

If you have multiple LDAP servers and want to use those for authentication, you can specify multiple LDAPInfo directives. Of course, each label must be unique (**1**).

When you have specified the LDAPInfo directive shown in Figure 16-28 on page 395 and the LDAP server is running, IBM HTTP Server tries to connect to the LDAP server at initialization time. If IBM HTTP Server succeeds in connecting to the LDAP server, you will see the following message in the trace obtained by specifying the -vv option in the server's start procedure:

```
Communications established successfully with LDAP servers.
```

If you do not have a -vv trace, you can check if IBM HTTP Server can connect to the LDAP server by using the **onetstat -c** command shown in the following:

```
TAKADA @ RA03:/u/takada>onetstat -p TCPIPA -c|grep WEBSRV
WEBSRV 00000C1D 172.16.250.3..1051 9.24.104.113..389 Establish
WEBSRV 00000C20 9.24.104.113..443 0.0.0.0..0 Listen
WEBSRV 00000C1F 9.24.104.113..80 0.0.0.0..0 Listen
```

16.9.2 How to use authentication information stored in LDAP

Figure 16-29 shows how to control access to resources with the label "CGI-Program" by using LDAP. You will see the new keyword "%%LDAP%%" in the PasswdFile and GroupFile subdirectives in this example. This means that IBM HTTP Server uses the LDAP directory for authentication.

```
Protection CGI-Program {
  ServerId      LDAP_Authentication 1
  PasswdFile    "%%LDAP:ITSOLDAPSRV%%" 2
  GroupFile     "%%LDAP:ITSOLDAPSRV%%" 3
  Mask         "cn=0S390 IP Security, ou=Groups, o=IBM_US, c=US" 4
}

Protect /cgi-bin/* CGI-Program INTERNAL
```

Figure 16-29 Using LDAP for authentication against protected resources

1 Specify the realm name. You can see the value you specified here in the window, which prompts users to enter their user IDs and passwords:

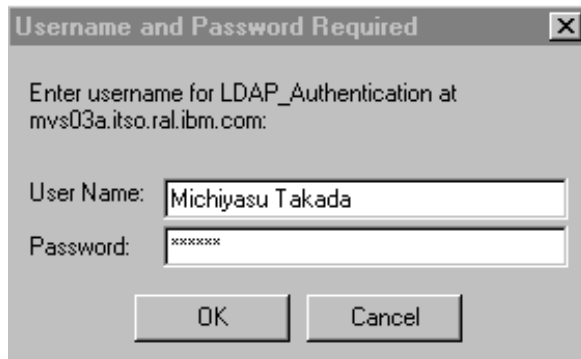


Figure 16-30 Netscape's window for entering a user name stored in LDAP

The user name entered in the window shown in Figure 16-30 is translated into "cn=Michiyasu Takada" based on the UserNameFilter specified in Figure 16-28 on page 395 (8). A password entered here must be registered in the LDAP directory.

2 This means that user IDs and passwords are stored in the LDAP directory. The syntax is `%%LDAP[:PrimaryLdapServer[, SecondaryLdapServer]]%%`. PrimaryLdapServer and SecondaryLdapServer match LDAP servers defined in LDAPInfo directives. The LDAPInfo directive must precede any `%%LDAP%%` references in the configuration file.

3 You can also specify the LDAP servers that you want to use for group information. The syntax is the same as that of PasswdFile. Even if the common name and password a user entered are correct, if that common name is not registered in the group entry specified in any mask subdirective (DeleteMask, GetMask, Mask, PostMask, and PutMask) and stored in the LDAP server specified in the GroupFile subdirective, IBM HTTP Server will deny this request.

4 This means that the common name a user entered must be contained in the group entry "cn=OS390 IP Security, ou=Groups, o=IBM_US, c=US" stored in the LDAP directory specified in the GroupFile directive.

Note: While it is possible to use LDAP for user authentication in IBM HTTP Server for z/OS, this option should be considered with great care. For each request for a resource protected by a Protection directive that specifies LDAP, the Web server needs to contact the LDAP server for authentication. For a large number of requests, this can create considerable overhead.

If RACF is used for authentication, z/OS UNIX will cache the security environments of users that have already been authenticated in main storage. Therefore, re-authentication of a user can be done with minimal overhead.

For all users that are defined in RACF, use `%%SAF%%` for authentication. LDAP can be considered for users that are not RACF defined but only defined in the LDAP directory. If the authentication frequency is high, methods for saving the authentication state of users should be considered to limit the overhead involved in LDAP authentication.

16.9.3 Creating user entries in the z/OS LDAP server

In this section, we show how to create the LDAP user entries and group entry. In this example, we used the z/OS LDAP Server. However, we do not show how to set up this server because that is beyond the scope of this book. If you want to know how to set up the z/OS LDAP server, see *z/OS V1R2.0 Security Server LDAP Server Administration and Use*, SC24-5923.

You can use the `ldif2db` utility to load entries from a file in LDAP Directory Interchange Format (LDIF) into a directory. The database must already exist and so must the suffixes under which new entries are being added. The `ldif2db` utility may be used to add entries to an empty directory database or to a database that already contains entries.

The syntax is:

```
ldif2db -i <input file> -f <configuration file>
```

No additional parameters other than the file names of the LDIF input file and the LDAP server configuration file are necessary since all other information (such as the suffix) is contained in the LDIF file.

The LDIF format is used to convey directory information or a description of a set of changes to directory entries. An LDIF file consists of a series of records separated by line separators. A record consists of a sequence of lines describing a directory entry or a sequence of lines describing a set of changes to a single directory entry. An LDIF file specifies a set of directory entries or a set of changes to be applied to directory entries but not both.

The following is an example of creating an input file in the LDIF format and then executing the `ldif2db` utility to load entries into the LDAP directory. We added four entries to our existing LDAP directory in order to use it for IBM HTTP Server authentication.

```
dn: cn=OS390 IP Security, ou=Groups, o=IBM_US, c=US2
objectclass: groupOfNames
description: OS/390 IP Security Team
cn: OS390 IP Security
member: cn=Otto Zech, ou=ITS0, ou=Raleigh, o=IBM_US, c=US
member: cn=Jorge Rivera, ou=ITS0, ou=Raleigh, o=IBM_US, c=US
member: cn=Michiyasu Takada, ou=ITS0, ou=Raleigh, o=IBM_US, c=US

dn: cn=Otto Zech, ou=ITS0, ou=Raleigh, o=IBM_US, c=US1
objectclass: organizationalPerson
cn: Otto Zech
sn: Zech
title: OS/390 UNIX System Services Security
userPassword: secret

dn: cn=Jorge Rivera, ou=ITS0, ou=Raleigh, o=IBM_US, c=US1
objectclass: organizationalPerson
cn: Jorge Rivera
sn: Rivera
title: OS/390 IPSec
userPassword: secret

dn: cn=Michiyasu Takada, ou=ITS0, ou=Raleigh, o=IBM_US, c=US1
objectclass: organizationalPerson
cn: Michiyasu Takada
sn: Takada
title: OS/390 UNIX Application Security
userPassword: secret
```

Figure 16-31 Input File in the LDIF format for creating the LDAP entries

¹ We created these entries as user accounts to access the resources with a label "CGI-Program" within IBM HTTP Server (see Figure 16-29 on page 396). When you are prompted to input your user ID and password, you have to input a common name (cn, for example, Otto Zech) and a password specified in the `userPassword` attribute. Then the

common name you entered is applied to the rule specified in the `UserNameFilter` subdirective, that is, "`(&(objectclass=organizationalPerson)(cn=%v1* %v2*))`". Finally, IBM HTTP Server looks for this entry in the LDAP directory based on the `UserSearchBase` subdirective (for example, "`o=IBM_US, c=US`") and checks if a password you entered corresponds to a value in the `userPassword` attribute.

2 In this example, we also specified the `%%LDAP%%` key word in the `GroupFile` subdirective (see Figure 16-29 on page 396). Therefore, we created a group entry permitted to access restricted resources. This distinguished name must coincide with one specified in the `Mask` subdirective. In addition, the user common name you entered from your browser must be included in this group entry.

```
TAKADA @ RA03:/u/takada/ldap>export STEPLIB=DB2V510.SDSNLOAD 1
TAKADA @ RA03:/u/takada/ldap>ldif2db -i ourteam.ldif -f /etc/ldap/slapd.conf 2
GLD0010I Reading configuration file /etc/ldap/slapd3.conf.
GLD0010I Reading configuration file /etc/ldap/slapd.at.system.
GLD0010I Reading configuration file /etc/ldap/slapd.cb.at.conf.
GLD0010I Reading configuration file /etc/ldap/slapd.at.conf.
GLD0010I Reading configuration file /etc/ldap/slapd.oc.system.
GLD0010I Reading configuration file /etc/ldap/slapd.cb.oc.conf.
GLD0010I Reading configuration file /etc/ldap/slapd.oc.conf.
GLD0010I Reading configuration file /etc/ldap/pagent_at.conf.
GLD0010I Reading configuration file /etc/ldap/pagent_oc.conf.
GLD0052I Configuration read securePort 636.
GLD0053I Configuration read security of none.
GLD0054I Value connectionsAllowed is set to 1.
GLD0040I The value for the 'maxthreads' option is out of range (10, 75000). The
default (75000) will be used.
GLD0129I The value for the 'maxConnections' option is out of range (0, 2047). Th
e default (2047) will be used.
GLD0130I The value for the 'waitingthreads' option is out of range (10, 75000).
The default (75000) will be used.
GLD2062E The LDAP Server will operate in single-server mode.
GLD2004I ldif2db: 4 entries have been successfully added out of 4 attempted. 3
```

Figure 16-32 Executing the `ldif2db` utility

Figure 16-32 shows how to execute the `ldif2db` utility. Before issuing the `ldif2db` utility, you have to define the DB2 load library in the `STEPLIB` environment variable if you have not specified the library to `LNKLST` (**1**). Then you can execute the `ldif2db` utility as shown in **2**. We used a file "`/u/takada/ldap/ourteam.ldif`" as input. If you see the message shown in **3**, you have succeeded in loading entries specified in an input file in the LDIF format.

Now you can use user entries stored in the LDAP directory for authentication. As mentioned earlier, these user entries can be used for authentication not only by IBM HTTP Server but also other LDAP-enabled applications such as WebSphere, Apache, and so on.

For more detailed information about LDAP support, refer to *z/OS V1R1.0-V1R3.0 IBM HTTP Server Planning, Installing, and Using V5.3*, SC34-4826.

16.10 Conclusion

These are our recommendations for using IBM HTTP Server for z/OS in a secure manner:

1. Intranet scenario

If users in the intranet access your company's common information such as company news, new product news, yellow pages and so forth, it is sufficient to create surrogate user IDs, for example, "INTERNAL". These user IDs should not have authority to access any resources beyond what is needed for the service they are supposed to provide. Consider defining these user IDs as RESTRICTED in RACF.

If you want to enable only certain users to access certain resources, specify the Protect and Protection directives (or the DefProt directive) in the server configuration file and instruct users to enter their RACF user ID.

When you let users access confidential information, such as employee pay data, evaluation data and so forth, you should encrypt this type of data using SSL. Furthermore, if possible, you should use client authentication and associate the client certificates with their RACF user IDs using the RACDCERT TSO command or the self-registration application. This enables users to access sensitive data securely without entering their passwords.

2. Internet scenario

If you want your Web server to be open to the public, you should create surrogate user IDs, for example, "EXTERNAL". Similar to the intranet case, these user IDs should have as little authority as possible. By creating surrogate user IDs, when users on the Internet want to access public information such as press releases and marketing information, they do not need passwords. On the other hand, if users need to transmit their confidential personal data such as credit card numbers, account numbers and so on, the connection between users and your Web server should be encrypted with SSL. The use of client certificates in an Internet environment requires careful planning, especially if large numbers of clients are involved. You should get your server certificate not by self-signing but from an external CA.



Utility applications

In this chapter we talk about the security considerations for what could be termed “utility” applications. The utility applications discussed in this chapter are LDAP (Lightweight Directory Access Protocol), DNS, and the syslogd daemon.

Lightweight Directory Access Protocol (LDAP) defines a standard method for accessing and updating information in a directory. LDAP is gaining wide acceptance as the directory access method of the Internet and is therefore also becoming strategic within corporate intranets. It is being supported by a growing number of software vendors and is being incorporated into a growing number of applications. For example, the two most popular Web browsers, Netscape Navigator/Communicator and Microsoft Internet Explorer, support LDAP as a base feature.

For a discussion on the use of Secure Sockets Layer (SSL) with the z/OS LDAP server, see 17.1, “z/OS Lightweight Directory Access Protocol (LDAP)” on page 402.

DNS is the Domain Name Server, a server used to convert host names to IP addresses and vice versa. Prior to z/OS V1R2 the DNS was based on BIND 4.9.3 (Berkeley Internet Name Domain - the original designers of DNS), but with z/OS V1R2, both BIND 4.9.3 and BIND 9 are supported. If you use DNS/WLM you must continue to use the BIND 4.9.3 based DNS, otherwise you can use the BIND 9 based DNS. There are two new security features in the BIND 9 based DNS, Transaction Signatures (TSIG) and DNS Security Extensions (DNSSEC).

The BIND-9 based DNS security features TSIG and DNSSEC are discussed in “BIND-9 based DNS” on page 409.

The syslog daemon accepts status and debugging messages from clients. These clients may be other servers or IP application programs. Syslogd can write the received messages to a range of destinations including the MVS console, log files, SMF, other machines, or users depending on the syslogd configuration file. Syslogd opens a listener on a UDP socket to enable other systems to send logging information. In order to prevent being flooded with messages from other systems, syslogd can be isolated.

For a discussion on isolating syslogd from other hosts systems, see 17.3, “Syslogd daemon” on page 419.

17.1 z/OS Lightweight Directory Access Protocol (LDAP)

In many organizations, a vast amount of information describing the various users, applications, files, printers, and other resources accessible from a network is often collected into a special database that is sometimes called a directory. As the number of different networks and applications has grown, the number of specialized directories of information has also grown, resulting in islands of information that are difficult to share and manage. If all of this information could be maintained and accessed in a consistent and controlled manner, it would provide a focal point for integrating a distributed environment into a consistent and seamless system. The Lightweight Directory Access Protocol (LDAP) is an open industry standard that has evolved to meet these needs.

Much critical security information such as security policies, user passwords, and X.509 digital certificates, is maintained in LDAP directories because LDAP is an open standard protocol supported by all the major vendors. This means an LDAP server is required to provide high availability, fault tolerance, efficient resource utilization, and high performance, not to mention a high degree of security for itself. If an LDAP server is down, it could mean that nobody can log on to anything.

The LDAP server is extremely beneficial in providing a centralized directory for (among other data) security information relating to users of your network. Because of this, the LDAP server itself requires a very high level of security protection.

On the z/OS platform, IBM HTTP Server exploits LDAP for authenticating requests from its clients and Policy Agent (PAGENT) uses the LDAP directory to store some policies for Quality of Service (QoS).

For information on the configuration and use of the LDAP server, see *Communications Server for z/OS V1R2 TCP/IP Implementation Guide Volume 6: Policy and Network Management*, SG24-6839.

This section covers the use of SSL to secure communications between an LDAP client and the z/OS LDAP server.

17.1.1 Authentication with the z/OS LDAP server

In many directories, a good deal of the information can be accessed by anonymous users without a need for authentication. However, there may be other information of a more confidential nature that can only be accessed by authorized users. Also, any update of data maintained by an LDAP server will usually be only allowed for authorized users.

Checking the authority of a user can only be useful if the user has been reliably authenticated. To fulfill this requirement, the LDAP server supports a number of authentication methods:

- ▶ Authentication with user name and password stored in the LDAP directory.
- ▶ Authentication with an X.509 V.3 client certificate.
- ▶ Authentication with the user's RACF user ID and password. The LDAP server will call RACF to authenticate the user's password.

The authentication of the client is performed when the LDAP client binds to the LDAP server. This is the process when the client opens a socket connection with the server.

If passwords are stored in the LDAP directory, a number of different methods are available to prevent unauthorized disclosure of passwords:

- ▶ One-way encryption using the SHA-1 or MD5 message digest.
These methods require the z/OS Open Cryptographic Services Facility (OCSF) to be installed. The clear-text password cannot be retrieved.
- ▶ One-way encryption using the crypt method.
Crypt is a method based on the DES encryption algorithm that is mostly used in UNIX systems to create one-way encrypted representations of passwords in the `/etc/passwd` file.
- ▶ Two-way encryption using DES.
This method requires the z/OS Open Cryptographic Services Facility (OCSF) to be installed. This is the only method that allows the clear-text password to be retrieved. If clear-text passwords need to be retrieved from the LDAP directory, this method should be used.

As an alternative to authentication with passwords, the LDAP server supports user authentication with X.509 V.3 client certificates. After verifying the authenticity of the user's certificate, the LDAP server will match the distinguished name (DN) of the certificate owner to the distinguished name of the user's directory entry. The session between the LDAP client and the LDAP server must be an SSL session, or client certificates cannot be used.

For users already defined in RACF, there is no incentive to create and store additional passwords in the LDAP directory since the users would be forced to maintain these passwords or synchronization methods would have to be developed. The LDAP server, if so configured, is able to take the user ID and password specified in the bind request and call RACF for authentication of the user. If the user ID and password supplied by the user are valid, the bind is successful.

This section discusses the client certificate authentication method, requiring SSL.

17.1.2 Security of the directory

Directories contain sensitive information that needs to be protected from unauthorized access and modification. When sending data over networks internally or externally, sensitive information may also need to be protected against eavesdropping and modification during transportation. There is a need to know who is requesting the information and who is sending it.

The z/OS LDAP Server provides:

- Authentication** The requester must prove his or her or its identity to the directory. This is supported using certificates using SASL/SSL.
- Access Control** The directory server only returns data that the requester is entitled to access. In other words, the requester must have adequate authorization. This is implemented through the use of access control lists (ACLs).
- Integrity** Data needs to be reliably stored and transmitted so that alterations can be detected. SSL network transmissions are protected against alterations.
- Confidentiality** Sensitive data transmitted to or from the directory or stored in the directory cannot be easily accessed without proper permission (authorization). User passwords can be stored encrypted in the directory. Network transmissions can be protected using SSL.

The LDAP server provides the following security mechanisms for authentication and privacy:

- ▶ Anonymous authentication

This is useful for read-only access of directory data where that data is not sensitive, such as e-mail addresses or office numbers. Essentially, that data is accessible to anyone. To request anonymous authentication, simple authentication is performed with a distinguished name (DN) that is empty.

- ▶ Basic authentication (distinguished name and password)

This provides authentication facilities with the DN and password transmitted over the network in clear text. The use of clear text passwords is not recommended over open networks when there is no authentication or encryption being performed by a lower layer, such as IPsec. Access (read or write) to directory data is granted based on DNs contained in the access control list of the object and/or attributes in the access request.

z/OS LDAP Server allows a user to be authenticated to the directory name space using the RACF user ID and password. The RACF identity becomes associated with the user's RACF-style distinguished name on the bind operation.

- ▶ Simple Authentication and Security Layer (SASL)

The Simple Authentication and Security Layer (SASL) is a framework for multiple authentication and encryption mechanisms for connection-oriented protocols, as described in RFC 2222. It has been added to LDAP Version 3 to overcome the authentication shortcomings of LDAP Version 2.

In SASL, connection protocols, such as LDAP, IMAP, and so on are represented by profiles; each profile is considered a protocol extension that allows the protocol and SASL to work together. Among these are IMAP4, SMTP, POP3, and LDAP. Each protocol that intends to use SASL needs to be extended with a command to identify an authentication mechanism and to carry out an authentication exchange. LDAP Version 3 includes such a command: `ldap_sasl_bind()`. Optionally, a security layer can be negotiated to encrypt the data after authentication and thus ensure confidentiality. The IBM SecureWay Directory running on Windows NT and AIX supports SASL authentication using the Challenge Response Authentication Mechanism with Message Digest 5 (CRAM-MD5) mechanism, which transmits message digests rather than the passwords themselves over the network.

In z/OS V1R2, the SASL bind mechanism of EXTERNAL is supported by the z/OS LDAP Server. This means that the authentication on the bind is performed by the SSL client authentication that was performed on the initial connection from the client. At the time of writing, OS/390 LDAP Server does not support SASL/CRAM-MD5 authentication.

- ▶ Secure Sockets Layer (SSL) certificate authentication and encryption

As discussed in 9.1, "Secure Sockets Layer (SSL)" on page 186, the Secure Sockets Layer (SSL) is, strictly speaking, not part of LDAP. It is a lower-level authentication and encryption service that higher-level applications, such as LDAP or HTTP, can use and rely on. It provides strong authentication using certificates and strong encryption using DES, Triple DES, RC2 or RC4 for securing network traffic. SSL is very widely used in the industry.

17.1.3 Using SSL communication

Much sensitive data, such as user passwords, X.509 certificates, private key, security policy and so forth, is stored in an LDAP directory. If a malicious user steals this information by using LAN analyzers or packet trace programs, not only the directory itself but also other systems or applications such as Web servers will be exposed to great danger. The key to protecting your data is to encrypt data and maintain data integrity. SSL is the de facto standard protocol, especially in HTTP communications, to achieve data encryption and data integrity.

The z/OS LDAP Server supports client authentication, with server authentication by the client mandatory.

For server authentication, the LDAP server must have a digital certificate (based on the X.509 standard) which it passes to the client in the initial SSL handshake messages. If the client validates the server's certificate, then a secure, encrypted communication channel is established between the LDAP server and the LDAP client.

If the LDAP server is configured for client authentication, and the client sends a digital certificate in the initial SSL handshake, it must be validated by the LDAP server before the secure encrypted communication channel is established between them. Client authentication by the LDAP server facilitates the use of the SASL bind mechanism of EXTERNAL by an LDAP client. The bind identity becomes the distinguished name in the client digital certificate.

Establishing server authentication

To establish SSL server authentication between the LDAP server and an LDAP client, you have to do the following:

- ▶ Create a server certificate and keyring. You can use the RACDCERT command to store the certificates in a RACF keyring or gskkyman if using an HFS.
- ▶ Create a key database for an LDAP client.
- ▶ Specify the SSL-related options in the LDAP configuration file.

In the following discussion, we show you how to establish the SSL environment. For further information about the certificate management on z/OS, refer to Chapter 10, "Certificate management in z/OS" on page 203.

Create a server certificate

We created a self-signed certificate for the z/OS LDAP server.

Key database file	/etc/ldap/ssl/ldapsrv.kdb
Key database password file	/etc/ldap/ssl/ldapsrv.sth
Certificate file	/etc/ldap/ssl/ldapsrv.crt

Though a self-signed certificate is useful for test purposes, if you want to open your directory to business partners or users on the Internet, you had better request a trusted CA to issue your server certificate.

Create a key database for an LDAP client

During the SSL handshake protocol, an LDAP client must ensure that a server certificate sent from the LDAP server was really issued by a trusted Certificate Authority. In this example, we are using an LDAP client on z/OS.

If you select a certificate by a trusted CA such as IBM Vault Registry, Thawte, RSA, or VeriSign, you only create a key database (if using gskkyman) because these CA's certificates are automatically included in any key database created by the gskkyman utility. If you are using RACDCERT, you must add the CA certificate to the keyring being used by the LDAP server.

On the other hand, if you create a self-signed certificate for the LDAP server, you need to register this certificate into a client key database as a trusted CA certificate.

Figure 17-1 on page 406 and Figure 17-2 on page 406 show how to create a key database using gskkyman for an LDAP client and store a self-signed server certificate into a client's key database.

```

TAKADA @ :/u/takada/ldap>gskkyman

          IBM Key Management Utility

Choose one of the following options to proceed.

    1 - Create new key database
    2 - Open key database
    3 - Change database password

    0 - Exit program

Enter your option number: 1
Enter key database name or press ENTER for "key.kdb": ldapclient.kdb1
Enter password for the key database.....>*****2
Enter password again for verification.....>*****
Should the password expire? (1 = yes, 0 = no) [1]: 1
Enter password expiration time (number of days) or press ENTER for 60 days:

The database has been successfully created, do you want to continue to work with
the database now? (1 = yes, 0 = no) [1]: 1

```

Figure 17-1 Creating a key database for an LDAP client

When you try to bind to z/OS LDAP Server using SSL, you must specify a key database name (1) and a password (2) to open this database.

```

          Key database menu

Current key database is /u/takada/ldap/ldapclient.kdb

    1 - List/Manage keys and certificates
    2 - List/Manage request keys
    3 - Create new key pair and certificate request
    4 - Receive a certificate issued for your request
    5 - Create a self-signed certificate
    6 - Store a CA certificate
    7 - Show the default key
    8 - Import keys
    9 - Export keys
    10 - List all trusted CAs
    11 - Store encrypted database password

    0 - Exit program

Enter option number (or press ENTER to return to the parent menu): 6
Enter certificate file name or press ENTER for "cert.arm": ldapsrv.crt
Enter a label for this key.....> LDAP Server Certificate

Please wait while certificate is stored...

Your request has completed successfully, exit gskkyman? (1 = yes, 0 = no) [0]:1

```

Figure 17-2 Storing a self-signed server certificate into a client's key database

Now this self-signed certificate has been registered as a trusted CA certificate in a key database for an LDAP client.

Specify the SSL-related options in the LDAP configuration file

Finally you have to set several SSL-related options in the LDAP server configuration file (normally `/etc/ldap/slapd.conf`):

```
securePort 636 1
security ssl 2
sslAuth serverAuth 3
sslKeyRingFile /etc/ldap/ssl/ldapsrv.kdb 4
sslKeyRingPWStashFile /etc/ldap/ssl/ldapsrv.st 5
sslCipherSpecs 12288 6
```

Where:

- 1** Specify a port used for SSL communications. The default is 636.
- 2** Set `ssl` in the `security` option. If you use only SSL communications, set `security sslonly`.
- 3** Specify `serverAuth` in the `sslAuth` option for server authentication. This is the default, and specifies that only the server is authenticated by SSL, not the client. (The client will be authenticated by the LDAP server.)
- 4** Specify a name of the server key database file you created before.
- 5** This is a password file to open a key database specified in **4**. The contents of this password file are encrypted. Instead of this option, you can set a password that you specified when you created a key database with the `sslKeyRingFilePW` option. However, you should not specify your password in a clear text because if malicious users find this configuration file, they can easily open the server key database specified here using this password.
- 6** This option is used to specify the SSL cipher specifications that will be accepted from clients. You can use the following ciphers for SSL communication between the LDAP server and LDAP clients only if the clients support these ciphers:

U.S. and Canada version:

- `SLAPD_SSL_RC4_MD5_US (0800)`
- `SLAPD_SSL_TRIPLE_DES_SHA_US (0100)`
- `SLAPD_SSL_DES_SHA_US (0200)`
- `SLAPD_SSL_RC2_MD5_EXPORT (1000)`
- `SLAPD_SSL_RC4_MD5_EXPORT (2000)`

Export version:

- `SLAPD_SSL_RC2_MD5_EXPORT (1000)`
- `SLAPD_SSL_RC4_MD5_EXPORT (2000)`

You need to specify an integer value with the `sslCipherSpecs` option. This integer is the decimal representation of the OR-ed bit mask defined by the hexadecimal values above.

In our example, we specified 12288 (hex 3000). In other words, we selected `RC2_MD5` and `RC4_MD5` as the SSL cipher specifications.

To make the LDAP server recognize these SSL-related options in the configuration file, you must recycle the server.

Test the SSL connection between LDAP client and server

```
TAKADA @ RA03:/u/takada/ldap>onetstat -p TCPIPA -c |grep LDAPSRV
LDAPSRV 0000129F 0.0.0.0..389          0.0.0.0..0          Listen
LDAPSRV 000012A0 0.0.0.0..636          0.0.0.0..0          Listen
```

Now you can establish SSL communication between z/OS LDAP Server and an LDAP client. Figure 17-3 shows an example of searching an LDAP entry through SSL using the `ldapsearch` command-line utility.

```
TAKADA @ RA03:/u/takada/ldap>cat ldapsearch.ssl.sh
ldapsearch -h 9.24.104.113 -Z -K ldapclient.kdb \ 1
  -P takada "cn=Michiyasu Takada"
TAKADA @ RA03:/u/takada/ldap>ldapsearch.ssl.sh
cn=Michiyasu Takada, ou=ITSO, ou=Raleigh, o=IBM_US, c=US
objectclass=organizationalPerson
cn=Michiyasu Takada
sn=Takada
telephonenumber=1-812-855-7743
title=OS/390 USS, LDAP, Firewall
postalcode=4609
```

Figure 17-3 Searching an LDAP entry through an SSL connection

The `-Z` option is used for a secure SSL connection to communicate with the LDAP Server. The `-Z` option is not supported by non-SSL versions of this tool.

The `-K` option is used to specify the name of the SSL key database file for an LDAP client. If the key database file is not in the current directory, specify the fully qualified key database file name. If a key database file name is not specified, this utility will look for the presence of the `SSL_KEYRING` environment variable with an associated file name. Otherwise, no key database file will be used for server authentication and the default trusted Certificate Authority roots will be used. This parameter is ignored if `-Z` is not specified.

The `-P` option is used to specify the key database file password. This password is required to access the encrypted information in the key database file (including the private key). If the key database file does not contain a private key, which is possible on the LDAP client, then the key database file may have been created without a password. In this case, there is no need to specify a password here. This parameter is ignored if `-Z` is not specified.

Similarly, you can use the other command-line utilities, such as `ldapadd`, `ldapmodify`, `ldapmodrdn`, `ldapdelete` and `ldapcp`, through the SSL communication channel.

For more detailed information about the LDAP command-line utility, see *z/OS SecureWay Security Server LDAP Server Administration and Use*, SC24-5923.

If you want to develop your own LDAP client application SSL-enabled, you need to add two extra function calls, `ldapssl_client_init()` and `ldapssl_init()`, to the source code of your application. For information on how to create your own SSL-enabled LDAP application, see *z/OS SecureWay Security Server LDAP Client Programming*, SC24-5924.

Establish client authentication

As discussed in 17.1.2, “Security of the directory” on page 403, the SASL bind mechanism of EXTERNAL is supported. If you select the SASL bind mechanism, the authentication on the bind is performed by the SSL client authentication.

To use SASL bind, the following steps must occur:

- ▶ The LDAP server must be configured and started with `sslAuth` set to `serverClientAuth` in the server configuration file so that the server can authenticate the client:

```
sslAuth serverClientAuth
```


- ▶ The client chooses SSL communication with the LDAP server through the use of `ldap_ssl_client_init` and `ldap_ssl_init`, and sends its certificate on `ldap_ssl_client_init` by having a client certificate with a private key and marking it as the default in its key database.
- ▶ The LDAP server authenticates the client's certificate (the LDAP server's key database contains and trusts the CA certificate of the signer of the client's certificate, or if the client is using a self-signed certificate the server contains and trusts the client's certificate).
 In our test environment, we created a self-signed certificate. Therefore we had to store an LDAP client's certificate in the server key database (`/etc/ldap/ssl/ldapsrv.kdb`) in the same way as shown in Figure 17-2 on page 406.
- ▶ The client performs a SASL bind with the LDAP server through the use of `ldap_sasl_bind` with the mechanism of EXTERNAL.

Again we examined the SSL client authentication using the same shell script shown in Figure 17-3 on page 408 and succeeded in searching an LDAP entry. At that time, we obtained the LDAP server trace to confirm if an LDAP client's certificate was really sent or not. The following is a part of the trace you can obtain by setting the `-d` flag in the LDAP server's started procedure:

```
Trying to extract client DN from CMS
SSLSupport_GetClientDN called
rc returned from CMS is 0
Client DN extracted from CMS is CN = LDAP Client, OU = ITS0, O = IBM_US,
L = Raleigh, stateOrProvinceName = North Carolina, C = US .
```

At this point, the LDAP server will consider the bind DN of the client for authorization purposes to be the client's DN as transmitted in the client's certificate on the `ldap_ssl_client_init`.

17.2 BIND-9 based DNS

This section is a summarization of the TSIG and DNSSEC features discussed in greater detail in *Communications Server for z/OS V1R2 TCP/IP Implementation Guide Volume 2: UNIX Applications*, SG24-5228.

17.2.1 TSIG

This is a short guide to setting up Transaction Signatures (TSIG) based transaction security in BIND. It describes changes to the configuration file as well as what changes are required for different features, including the process of creating transaction keys and using transaction signatures with BIND. BIND primarily supports TSIG for server-to-server communication. This includes zone transfer, notify, and recursive query messages. Resolvers on other platforms that are based on newer versions of BIND 8 have limited support for TSIG.

TSIG might be most useful for dynamic update. A master server for a dynamic zone should use access control to control updates, but IP-based access control is insufficient. Key-based access control is far superior. The `nsupdate` command supports TSIG via the `-k` and `-y` command-line options.

To implement TSIG in your environment, follow these steps:

1. Generate shared keys for each pair of hosts
2. Copy the shared secret to both machines
3. Inform the servers of the key's existence
4. Instruct the server to use the key

5. TSIG key-based access control
6. Resolve errors

Generate shared keys for each pair of hosts

A shared secret is generated to be shared between host1 and host2. An arbitrary key name is chosen: "host1-host2.". The key name must be the same on both hosts. The shared secret can be generated either automatically or manually, as follows:

Automatic Generation: The following command will generate a 128-bit (16-byte) HMAC-MD5 key as described above. Longer keys are better, but shorter keys are easier to read. Note that the maximum key length is 512 bits; keys longer than that will be digested with MD5 to produce a 128 bit key.

```
dnssec-keygen -a hmac-md5 -b 128 -n HOST host1-host2.
```

The key is in the file `Khost1-host2.+157+00000.private`. Nothing directly uses this file, but the Base64 encoded string following "Key:" can be extracted from the file and used as a shared secret:

```
Key: La/E5CjG90+os1jq0a2jdA==
```

The string "La/E5CjG90+os1jq0a2jdA==" can be used as the shared secret.

Manual Generation: The shared secret is simply a random sequence of bits, encoded in Base64. Most EBCDIC strings are valid Base64 strings (assuming the length is a multiple of 4 and only valid characters are used), so the shared secret can be manually generated. Also, a known string can be run through `mmencode` or a similar program to generate Base64 encoded data.

Copy the shared secret to both machines

This is beyond the scope of DNS. A secure transport mechanism should be used. This could be secure FTP, ssh, telephone, etc.

Inform the servers of the key's existence

Imagine host1 and host2 are both servers. The following is added to each server's `named.conf` file:

```
key host1-host2. {
    algorithm hmac-md5;
    secret "La/E5CjG90+os1jq0a2jdA=";
};
```

The algorithm, `hmac-md5`, is the only one supported by BIND. The secret is the one generated above. Since this is a secret, it is recommended that either `named.conf` be non-world readable, or the key directive be added to a non-world readable file that is included by `named.conf`. At this point, the key is recognized. This means that if the server receives a message signed by this key, it can verify the signature. If the signature succeeds, the response is signed by the same key.

Instruct the server to use the key

Since keys are shared between two hosts only, the server must be told when keys are to be used. The following is added to the `named.conf` file for host1, if the IP address of host2 is 10.1.2.3:

```
server 10.1.2.3 {
    keys { host1-host2. };
};
```

Multiple keys may be present, but only the first is used. This directive does not contain any secrets, so it may be in a world-readable file. If host1 sends a message that is a request to that address, the message will be signed with the specified key. host1 will expect any responses to signed messages to be signed with the same key. A similar statement must be present in host2's configuration file (with host1's address) for host2 to sign request messages to host1.

TSIG key-based access control

BIND allows IP addresses and ranges to be specified in ACL definitions and `allow-{ query | transfer | update }` directives. This has been extended to allow TSIG keys also. The above key would be denoted `key host1-host2`. An example of an `allow-update` directive would be:

```
allow-update { key host1-host2. ;};
```

This allows dynamic updates to succeed only if the request was signed by a key named "host1-host2".

Errors

The processing of TSIG signed messages can result in several errors. If a signed message is sent to a non-TSIG aware server, a FORMERR will be returned, since the server will not understand the record. This is a result of misconfiguration, since the server must be explicitly configured to send a TSIG signed message to a specific server. If a TSIG-aware server receives a message signed by an unknown key, the response will be unsigned with the TSIG extended error code set to BADKEY. If a TSIG-aware server receives a message with a signature that does not validate, the response will be unsigned with the TSIG extended error code set to BADSIG. If a TSIG-aware server receives a message with a time outside of the allowed range, the response will be signed with the TSIG extended error code set to BADTIME, and the time values will be adjusted so that the response can be successfully verified. In any of these cases, the message's response code is set to NOTAUTH.

17.2.2 DNSSEC

Cryptographic authentication of DNS information is possible through the DNS Security (DNSSEC) extensions, defined in RFC 2535. This section describes the creation and use of DNSSEC signed zones.

The set of `dnssec-` tools rely on a `/dev/random` device for the entropy it needs to generate cryptographically strong keys. If RSA keys are used, only `dnssec-keygen` requires random data. z/OS UNIX does not include such a device, but the tools provide alternative methods of providing them with random data. The user can specify a file containing random data or can provide random data via the keyboard. To specify a file, use the `-r random data file` option on the tool command line. The `dnssec-` tools use the timing between keystrokes as the source of entropy. As such, TN3270 terminal emulation is not the ideal interface. Setting up a VT100 terminal session is a better solution. Refer to *Communications Server for z/OS V1R2 TCP/IP Implementation Guide Volume 2: UNIX Applications*, SG24-5228 for more information on setting up `otelnetd`.

In order to set up a DNSSEC secure zone, there are a series of steps that must be followed. z/OS ships with several tools that are used in this process, which are explained in more detail in the DNS chapter of *Communications Server for z/OS V1R2 TCP/IP Implementation Guide Volume 2: UNIX Applications*, SG24-5228. In all cases, the `-h` option prints a full list of parameters. Note that the DNSSEC tools require the `keyset` and `signedkey` files to be in the working directory. There must also be communication with the administrators of the parent

and/or child zone to transmit keys and signatures. A zone's security status must be indicated by the parent zone for a DNSSEC-capable resolver to trust its data. For other servers to trust data in this zone, they must either be statically configured with this zone's zone key or the zone key of another zone above this one in the DNS tree using the trusted-keys statement.

To implement DNSSEC, follow these steps:

1. Generate keys
2. Create a keyset
3. Sign the child's keyset
4. Sign the zone
5. Configure servers

Generate keys

The **dnssec-keygen** command is used to generate keys. A secure zone must contain one or more zone keys. The zone keys will sign all other records in the zone, as well as the zone keys of any secure delegated zones. Zone keys must have the same name as the zone, a name type of ZONE, and must be usable for authentication. The following command will generate a 768-bit RSA key for the child.example zone:

```
dnssec-keygen -a RSA -b 768 -n ZONE child.example
```

Two output files will be produced: Kchild.example.+001+12345.key and Kchild.example.+001+12345.private (where 12345 is an example of a key tag). The key file names contain the key name (child.example.), algorithm (3 is DSA, 1 is RSA, etc.), and the key tag (12345 in this case). The private key (in the private file) is used to generate signatures, and the public key (in the .key file) is used for signature verification. To generate another key with the same properties (but with a different key tag), repeat the above command. The public keys should be inserted into the zone file with \$INCLUDE statements, including the .key files.

Create a keyset

The **dnssec-makekeyset** command is used to create a keyset from one or more keys. Once the zone keys have been generated, a keyset must be built for transmission to the administrator of the parent zone, so that the parent zone can sign the keys with its own zone key and correctly indicate the security status of this zone. When building a keyset, the list of keys to be included and the TTL of the set must be specified, and the desired signature validity period of the parent's signature may also be specified. The list of keys to be inserted into the keyset may also include non-zone keys present at the top of the zone.

dnssec-makekeyset may also be used at other names in the zone.

The following command generates a keyset containing the above key and another key similarly generated, with a TTL of 3600 and a signature validity period of 10 days starting from now.

```
dnssec-makekeyset -t 3600 -e +8640 Kchild.example.+001+12345  
Kchild.example.+001+23456
```

One output file is produced: keyset-child.example. This file should be transmitted to the parent to be signed. It includes the keys, as well as signatures over the keyset generated by the zone keys themselves, which are used to prove ownership of the private keys and encode the desired validity period.

Sign the child's keyset

The `dnssec-signkey` command is used to sign one child's keyset. If the `child.example` zone has any delegations that are secure (for example, `grand.child.example`), the `child.example` administrator should receive keyset files for each secure subzone. These keys must be signed by this zone's zone keys. The following command signs the child's keyset with the zone keys:

```
dnssec-signkey keyset-grand.child.example. Kchild.example.+001+12345
Kchild.example.+001+23456
```

One output file is produced: `signedkey-grand.child.example`. This file should be both transmitted back to the child and retained. It includes all keys (the child's keys) from the keyset file and signatures generated by this zone's zone keys.

Sign the zone

The `dnssec-signzone` command is used to sign a zone. Any `signedkey` files corresponding to secure subzones should be present, as well as a `signedkey` file for this zone generated by the parent (if there is one). The zone signer will generate NXT and SIG records for the zone, as well as incorporate the zone key signature from the parent and indicate the security status at all delegation points. The following command signs the zone, assuming it is in a file called `zone.child.example`. By default, all zone keys that have an available private key are used to generate signatures.

```
dnssec-signzone -o child.example zone.child.example
```

One output file is produced: `zone.child.example.signed`. This file should be referenced by `named.conf` as the input file for the zone.

Configure servers

Data is not verified on load in BIND 9, so zone keys for authoritative zones do not need to be specified in the configuration file. The public key for any security root must be present in the configuration file's `trusted-keys` statement.

17.2.3 Secure your DNS environment

This is a short implementation scenario using TSIG and DNSSEC. For a more thorough treatment, see *Communications Server for z/OS V1R2 TCP/IP Implementation Guide Volume 2: UNIX Applications*, SG24-5228. The security implementation scenario will follow these steps:

1. Create a transaction signature between master and slave
2. Sign your zone

Create a transaction signature between master and slave

This step uses TSIG to let you restrict zone transfers to slave name servers that include a correct transaction signature with their request. On the master name server, generate a key, `dns63-dns64`, using the command `dnssec-keygen` as follows:

```
OCTAVIO @ SC63: />dnssec-keygen -a hmac-md5 -b 128 -n HOST dns63-dns64
start typing:
.....
.....
stop typing.
Kdns63-dns64.+157+37860
```

This generates two key files:

```
Kdns63-dns64.+157+37860.key  
Kdns63-dns64.+157+37860.private
```

The contents of the latter file is shown in Figure 17-4:

```
Private-key-format: v1.2  
Algorithm: 157 (HMAC_MD5)  
Key: /IMUHzLp15zem0t5JPdprw==
```

Figure 17-4 Contents of file *Kdns63-dns64.+157+37860.private*

Get the key you generated, insert it in a key statement and then specify the key in the address match list as follows:

```
key dns63-dns64 {  
    algorithm "hmac-md5";  
    secret "/IMUHzLp15zem0t5JPdprw==";  
};  
zone "zos12.ral.ibm.com" in {  
    type master;  
    file "zos12.for.v9";  
    allow-query { "zos12-net"; };  
    allow-transfer { key dns63-dns64.; };  
};
```

On the slave's end, you need to configure the slave to sign zone transfer requests with the same key:

```
key sc63-sc64. {  
    algorithm hmac-md5;  
    secret "/IMUHzLp15zem0t5JPdprw=";  
};  
  
server 9.12.6.68 {  
    keys { sc63-sc64.; };  
};  
  
zone "zos12.ral.ibm.com" {  
    type slave;  
    masters { 9.12.6.68; };  
    file "zos12.for.bak.v9";  
};
```

For a primary master name server accessible from the Internet, you probably want to limit zone transfers to just your slave name servers. You probably do not need to worry about name servers inside your firewall, unless you're worried about your own employees listing your zone data.

Sign your zone

The last and higher level of security to implement in our DNS environment is to use DNSSEC to sign your zone. We'll show you how we signed our DNS scenario, which is the zone called *zos12.ral.ibm.com*. All process will be done using the BIND 9 tools.

Generate your key pair

First, we generated a key pair for zone `zos12.ral.ibm.com` using `dnssec-key`:

```
OCTAVIO @ SC63:>dnssec-keygen -a DSA -b 512 -n ZONE zos12.ral.ibm.com
start typing:
.....
.....
stop typing.
Kzos12.ral.ibm.com.+003+09520
```

We ran `dnssec-keygen` in our name server's working directory. That's mostly for convenience; the zone data files are in this directory, so we won't need to use full pathnames as arguments. If we want to use dynamic update with DNSSEC, however, we'd need the keys in the name server's working directory.

Recall `dnssec-keygen`'s options:

- a The cryptographic algorithm to use, in this case RSA. We could also have used DSA, but RSA is more efficient.
- b The length of the keys to generate, in bits. RSA keys can be anywhere from 512 to 2000 bits long. DSA keys can be 512 to 1024 bits long, as long as the length is divisible by 64.
- n The type of key. DNSSEC keys are always ZONE keys.

The only non-option argument is the domain name of the zone, `zos12.ral.ibm.com`. The `dnssec-keygen` program prints the basename of the files it's written the keys to. The numbers at the end of the basename (`+003` and `+09520`) are the key's DNSSEC algorithm number as used in the KEY record (`003` is DSA/MD5), and the key's fingerprint (`+09520`), used to distinguish one key from another when multiple keys are associated with the same zone.

The public key is written to the file `basename.key` (`Kzos12.ral.ibm.com.+003+09520.key`). The private key is written to the `basename.private` (`Kzos12.ral.ibm.com.+003+09520.key`) file. Remember to protect the private key; anyone who knows the private key can forge signed zone data. `dnssec-keygen` does what it can to help you: it makes the `.private` file readable and writable only by the user who runs it.

Send your keys to be signed (optional)

Next, we sent our KEY record to the administrator of our parent zone to sign. BIND 9 includes a program to package up the key for transmission, `dnssec-makekeyset`:

```
# > dnssec-makekeyset -t 172800 Kzos12.ral.ibm.com.+003+09520.key
```

The `dnssec-makekeyset` command created a file called `keyset-zos12.ral.ibm.com`. Its contents are shown in Figure 17-5 on page 416.

```

$ORIGIN .
$TTL 172800 ; 2 days
zos12.ra1.ibm.com IN KEY 256 3 3 (
    AILb05yIGFZAsGCeh67DNVh7cYytzwmtkRbEERPh9rWy
    M1QbHuUBXTHdU/wsIvBr7omnRUXvLQxXEF4mLUnILL7j
    G5yncdRBo1KR7XTDkzGI9CW4qU4UGpH98QnbnZupGXDw
    nrHs8pK7stgssfjniShyUAbzVVnSxCtCHC7EQbnvIKdf
    jI5pydYKFeooXBd02kGPKmQinZfVdROKqstD5jFZ/+lh
    TjUKTZGYFDc57i6yMPNErsP8DI3GwQ410NaP20tCGMTJ
    YtIOhb9oNkU0tS9fgtIP ) ; key id = 10547
SIG KEY 3 4 172800 20020626173942 (
    20020527173942 9520 zos12.ra1.ibm.com.
    AHUWn73ryAc3TInk80lCso/1Mb3QF02026Xop+8nibcK
    WrtY24+DJsE= )

```

Figure 17-5 Contents of keyset-zos12.ra1.ibm.com. file

The -t option takes a TTL for the records to submit. This serves as a suggestion to your parent zone's administrator of the TTL (in seconds) you'd like for your record. They may ignore it, of course. The SIG record actually contains a signature covering your zone's KEY record, generated with your zone's private key. That proves you really have the private key that corresponds to the public key in the KEY record - you're not just submitting a KEY record you found on the street.

Then we sent our file off to our parent zone's administrators to sign. Since the message included proof of our identity, they signed it with the dnssec-signkey program:

```
# dnssec-signkey keyset-zos12.ra1.ibm.com Kral.ibm.com.+003+64185.private
```

Then they sent the resulting file, signedkey-zos12.ra1.ibm.com, back to us. Its contents are shown in Figure 17-6.

```

$ORIGIN .
$TTL 172800 ; 2 days
zos12.ra1.ibm.com IN KEY 256 3 3 (
    AILb05yIGFZAsGCeh67DNVh7cYytzwmtkRbEERPh9rWy
    M1QbHuUBXTHdU/wsIvBr7omnRUXvLQxXEF4mLUnILL7j
    G5yncdRBo1KR7XTDkzGI9CW4qU4UGpH98QnbnZupGXDw
    nrHs8pK7stgssfjniShyUAbzVVnSxCtCHC7EQbnvIKdf
    jI5pydYKFeooXBd02kGPKmQinZfVdROKqstD5jFZ/+lh
    TjUKTZGYFDc57i6yMPNErsP8DI3GwQ410NaP20tCGMTJ
    YtIOhb9oNkU0tS9fgtIP ) ; key id = 10547
SIG KEY 3 4 172800 20020626173942 (
    20020527173942 64185 ra1.ibm.com.
    AEvxWeXonAFwpaQp9nYhI5GDqafchSTb1wL1oAh+841m
    4FZzImMpJF4= )

```

Figure 17-6 signedkey-zos12.ra1.ibm.com. contents

If we didn't care about getting our KEY record signed, we could have skipped this step. But then only name servers with a trusted-keys entry for zos.ra1.ibm.com could verify our data.

Sign your zone

Before signing our zone, we had to include a reference to the key file into the our zone data file (zos12.for.v9) using the \$INCLUDE statement:

```
$INCLUDE Kzos12.ra1.ibm.com.+003+09520.key
```


That gave the signer program the information it needed to know which key to use to sign the zone. It automatically finds and includes the contents of `signedkey-zos12.ral.ibm.com`.

Then we signed the zone with `dnssec-signzone`:

```
# dnssec-signzone -o zos12.ral.ibm.com. zos12.for.v9
```

We used the `-o` option to specify the origin in the zone data file, because `dnssec-signzone` doesn't read `named.conf` to determine which zone the file describes. The only non-option argument is the name of the zone data file.

This produces a new zone data file, `zos12.for.v9.signed`. Partial contents of this file are shown in Figure 17-7.

```
; File written on Mon May 27 14:26:46 2002
; dnssec_signzone version 9.1.1
zos12.ral.ibm.com. 86400 IN SOA dns63.zos12.ral.ibm.com. admin.zos12.ral.ibm.com. (
    4          ; serial
    10800     ; refresh (3 hours)
    3600      ; retry (1 hour)
    604800    ; expire (1 week)
    86400     ; minimum (1 day)
)
86400 SIG SOA 3 4 86400 20020626182646 (
    20020527182646 9520 zos12.ral.ibm.com.
    AF2UFK1AP5eTS1pvgayWUN84rvREMzM3paa3
    /KJwdJ/EZ0vnAuIAuc4= )
86400 NS dns63.zos12.ral.ibm.com.
86400 NS dns63.zos12.ral.ibm.com.
86400 NS dns64.zos12.ral.ibm.com.
86400 SIG NS 3 4 86400 20020626182646 (
    20020527182646 9520 zos12.ral.ibm.com.
    AH1qWnUz5L9xVKBE0c6twGTXyp6BKLPtNRJo
    PEhuc2GXBvgHhYwssj4= )
86400 KEY 256 3 3 (
    AILb05yIGFZAsGCeh67DNVh7cYytzwmkRbE
    ERPh9rWyMTIqbHuUBXTHdU/wsIvBr7omnRUXv
    LQxXEF4mLUnILL7jG5yncdRBo1KR7XTDkzGI
    9CW4qU4UGpH98QnbnZupGXDwnrHs8pK7stgs
    sfjniShyUAbzVVnSxCtCHC7EQbnvIKdfjI5p
    ydYKFeooXBd02kGPKmQinZfVdROKqstD5jFZ
    /+1hTjUKTZGYFDc57i6yMPNErsP8DI3GwQ41
    ONaP20tCGMTJYtI0hb9oNkU0tS9fgtIP ) ; key id = 10547
172800 SIG KEY 3 4 172800 20020626173942 (
    20020527173942 64185 ral.ibm.com.
    AEVxWeXonAFwpaQp9nYhI5GDqafCHSTb1wL1
    oAh+841m4FZzImMpJF4= )
```

Figure 17-7 Zone data file created by command `dnssec-signzon`

Finally, we changed the zone statement in `named.conf` so that it would load the new zone data file:

```
zone "zos12.ral.ibm.com" in {
    type master;
    file "zos12.for.v9.signed";
    allow-query { "zos12-net"; };
    allow-transfer { key dns63-dns64.; };
    allow-update { 9.12.6.67;9.12.6.68;9.12.6.63; };
};
```

Then we reloaded the zone using `rndc reload` command and checked syslog.

DNSSEC and dynamic update

`dnssec-signzone` isn't the only way to sign zone data. The BIND 9 name server is capable of signing dynamically updated records. As long as the private key for a secure zone is available in the name server's working directory (in the correctly named `.private` file), a BIND 9 name server signs any records that are added via dynamic update. If any records are added to or deleted from the zone, the name server adjusts (and re-signs) the neighboring NXT records, too. The next steps show how it is done:

- ▶ Query the name server about a host called `newhost` using the `dig` command:

```
/> dig +dnssec newhost.zos12.ral.ibm.com.
```

The result of the command will be (we trimmed the output a little):

```
OCTAVIO@SC63:/>dig +dnssec newhost.zos12.ral.ibm.com
Allocated socket 5, type udp

; <<>> DiG 9.1.1 <<>> +dnssec newhost.zos12.ral.ibm.com
;; global options: printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NXDOMAIN, id: 49597
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 0, AUTHORITY: 4, ADDITIONAL: 3

;; OPT PSEUDOSECTION:
; EDNS: version: 0, udp= 4096
;; QUESTION SECTION:
;newhost.zos12.ral.ibm.com.      IN      A

;; AUTHORITY SECTION:
zos12.ral.ibm.com.      86400  IN      SOA     dns63.zos12.ral.ibm.com.
admin.zos12.ral.ibm.com.
zos12.ral.ibm.com.      86400  IN      SIG     SOA 3 4 86400 20020626182646
20020527182646 9520 z
gayWUN84rvREMzM3paa3/KJwdJ/EZ0vnAuIAuc4=
mail.zos12.ral.ibm.com. 86400  IN      NXT    sc63.zos12.ral.ibm.com. CNAME SIG NXT
mail.zos12.ral.ibm.com. 86400  IN      SIG     NXT 3 5 86400 20020626182646
20020527182646 9520 z
;; Query time: 2 msec
;; SERVER: 9.12.6.68#53(9.12.6.68)
;; WHEN: Mon May 27 15:23:29 2002
;; MSG SIZE rcvd: 640
```

Notice `mail.zos12.ral.ibm.com` NXT record, which is the last record in our domain file, indicating that the domain name doesn't exist. Now we'll use `nsupdate` to add an address record for `newzone.zos12.ral.ibm.com`:

```
/>nsupdate -V v9
> update add newhost.zos12.ral.ibm.com. 3600 IN A 9.12.6.61
```

Now, let's look up `newhost.zos12.ral.ibm.com`:

```
% dig +dnssec newhost.zos12.ral.ibm.com.
OCTAVIO@SC63:/>dig +dnssec newhost.zos12.ral.ibm.com
Allocated socket 5, type udp

; <<>> DiG 9.1.1 <<>> +dnssec newhost.zos12.ral.ibm.com
;; global options: printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 11973
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 4, ADDITIONAL: 9
```

```

;; QUESTION SECTION:
newhost.zos12.ra1.ibm.com.      IN      A
;; ANSWER SECTION:
newhost.zos12.ra1.ibm.com. 3600    IN      A      9.12.6.61
newhost.zos12.ra1.ibm.com. 3600    IN      SIG     A 1 3 3600 20010215195456
20010116185456 27791 zos12.ra1.ibm.com.
C/JXdCLUdugxN91v0DZuUDTusi2XNNttb4bdB2nBujLxjwwPAf/D5MJz
//cDtuZ3X+uYzhkN8MDR0q0wUQuQSA==

;; AUTHORITY SECTION:
zos12.ra1.ibm.com.      86400   IN      SOA     dns63.zos12.ra1.ibm.com.
admin.zos12.ra1.ibm.com.
zos12.ra1.ibm.com.      86400   IN      SIG     SOA 3 4 86400 20020626182646
20020527182646 9520 z
gayWUN84rvREMz3paa3/KJwdJ/EZ0vnAuIAuc4=
mail.zos12.ra1.ibm.com. 86400   IN      NXT     sc63.zos12.ra1.ibm.com. CNAME SIG NXT
mail.zos12.ra1.ibm.com. 86400   IN      SIG     NXT 3 5 86400 20020626182646
20020527182646 9520 z
;; Query time: 2 msec
;; SERVER: 9.12.6.68#53(9.12.6.68)
;; WHEN: Mon May 27 15:23:29 2002
;; MSG SIZE rcvd: 640

```

(Again, we trimmed the output a little.) Now not only was an address record generated, but there is also a SIG record generated from movie.edu's private key. As impressive as this is, you should be careful when allowing dynamic updates to secure zones. You should make sure that you use strong authentication (for example, TSIG) to authenticate the updates, or you'll give a hacker an easy back door to use to modify your "secure" zone.

And you should ensure you have enough horsepower for the task: normally, dynamic updates don't take much to process. But dynamic updates to a secure zone require NXT recalculation and, more significantly, asymmetric encryption (to calculate new SIG records), so you should expect your name server to take longer and need more resources to process them.

This ends the short scenario for TSIG and DNSSEC. More information can be found in *Communications Server for z/OS V1R2 TCP/IP Implementation Guide Volume 2: UNIX Applications*, SG24-5228.

17.3 Syslogd daemon

The syslog daemon reads and logs system messages to the MVS console, log files, SMF, other machines, or users. If syslogd is not started, application data may appear on the MVS console.

You may wish to have remote systems from remote, critical UNIX servers, send their log messages to the syslog daemon on z/OS, if there are important messages that you need to monitor from the mainframe.

Syslog facility names are specified in `/etc/syslog.conf` with associated log files that are stored in the Hierarchical File System (HFS). Depending on how the application has been coded, debug messages may go to one file and trace messages to another. The documentation for each server should indicate where the log debug messages, error messages, information messages and warning messages should be written. Some will log all of them to the same syslog destination.

syslogd may be configured either to forward messages to another syslogd on another host instead of logging them to files, to send them directly to a specified user or to all users that are logged in on the system, or to send messages to the MVS console.

Note in Figure 17-8 how the syslogd process creates a UDP socket and binds to port 514 during initialization. This port is reserved to OMVS in the z/OS TCP/IP stack and is also set aside in the /etc/services file. Local processes create an AF_UNIX socket to communicate with syslogd for logging purposes. An AF_INET socket is used for logging by and to remote servers. Herein lies a security exposure.

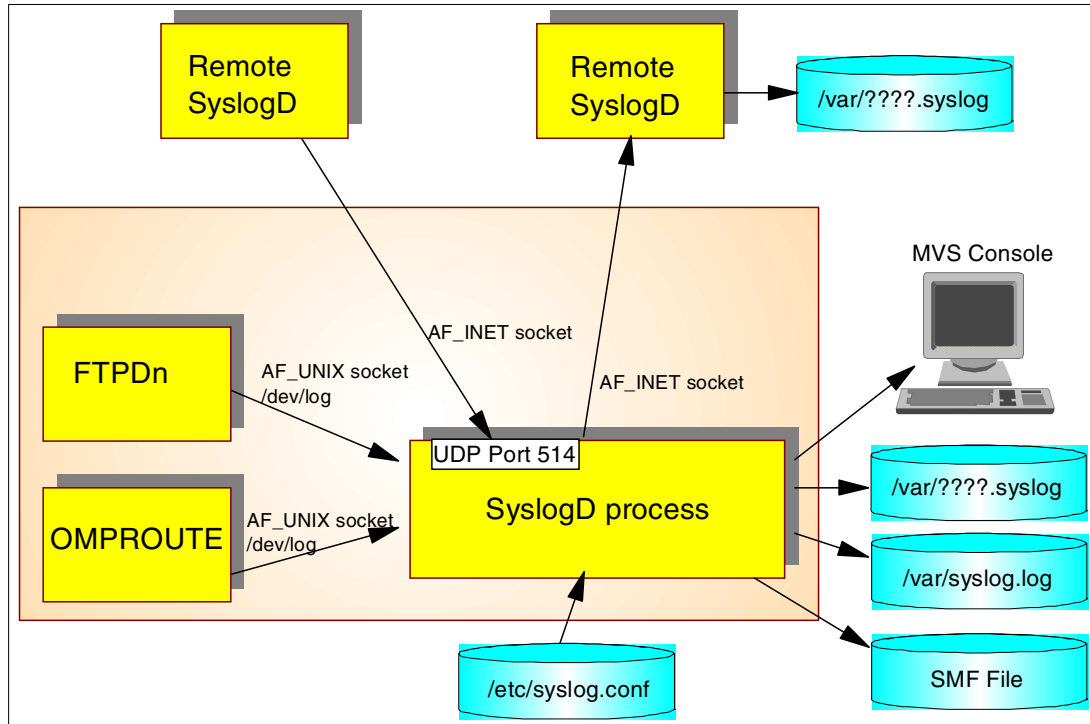


Figure 17-8 Syslog daemon processing

You may not wish to have remote syslogd clients and servers log into your log files. Log files can grow notoriously large and impact performance on your system. Furthermore, should the log files fill up the HFS designated for your logging, all further logging is stopped and you may lose critical information about system activity. Remote syslogd users may represent a security threat to your system, either because they overload the system with messages or because they send bogus messages that cause disruption to the system operation.

17.3.1 syslogd isolation

Syslogd isolation refers to the capability of syslogd to isolate messages into separate logs with more granularity than was previously possible and to restrict communication with remote syslog servers and clients.

In this section we discuss the aspect of syslogd isolation that restricts communication with other hosts. For more information on any other aspect of syslogd, see *z/OS V1R2.0 CS: IP Configuration Reference*, SC31-8776 or *z/OS V1R2.0 CS: IP Configuration Guide*, SC31-8775.

If you intend to receive log data from or send log data to remote syslogd servers, you should place the syslog service in the services definition file. Prior to CS for OS/390 V2R10 IP, logging by remote syslogd servers was prevented by eliminating this reference to the syslog service at UDP port 514 in the /etc/services file. However, the omission of a service reference for syslogd also prevented z/OS from logging messages at remote syslogd servers. The behavior of syslogd with respect to the /etc/services file changes in V2R10 and a new startup option, -i, is introduced to control the receipt of messages from remote syslog daemons.

Note the options that may be specified when initializing syslogd on z/OS:

```
syslogd [-f conffile] [-i] [-u[-c[-d]] [-m interval] [-p logpath]
```

As you can see from Table 17-1, only the -i option is used to control communication with other IP hosts.

Table 17-1 syslogd startup options

Option	Option Description
-c	Create log files and directories automatically.
-d	Run syslogd in debugging mode.
-f	Configuration file name.
-i	Do not receive messages from the IP network.
-m	Number of minutes between mark messages. The default value is 20 minutes.
-p	Path name of z/OS UNIX character device for the AF_UNIX datagram socket. The default value is /dev/log. This option is not used frequently. If you incorrectly use the -p option, syslogd will not function properly.
-u	For records received over the AF_UNIX socket (most messages generated on the local system), include the user ID and job name in the record.

You can configure syslogd on z/OS to log at remote servers while prohibiting the remote servers to log at z/OS itself. The -i command-line option at syslogd initialization tells syslogd not to receive messages via UDP (that is, from syslog daemons on other hosts in the network). So, with -i, syslogd is still able to send messages to other systems, as in the following configuration file example:

```
*.panic @bigiron.raleigh.ibm.com
```

If -i has been specified at syslogd startup and no remote destinations are named in the syslog.conf file, syslogd will not make use of AF_INET sockets at all.

Table 17-2 shows which ports are bound to under various startup conditions for syslogd.

Table 17-2 syslogd behavior with/without -i startup option

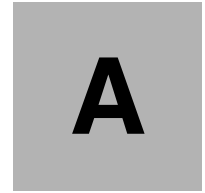
	Startup with -i	Startup without -i
Remote host configured in /etc/syslog.conf	syslogd binds to ephemeral port, not to udp/514	syslogd binds to port udp/514
Remote host not configured in /etc/syslog.conf	syslogd does not bind to any port, neither an ephemeral port nor udp/514	syslogd binds to port udp/514



Part 5

Appendixes

This final part contains some useful reference information relevant to this book. Please refer to the appropriate chapters for additional information.



VPN planning worksheets

Using the following templates, specify the information needed to plan for your dynamic VPN configuration. Create as many worksheets as you need for each TCP/IP stack planned to be configured with a dynamic tunnel.

Table A-1 Initial design worksheet for dynamic connections

Information needed to create dynamic VPNs	Answers
<p>What is the type of connection to be created?</p> <ul style="list-style-type: none"> - Gateway-to-gateway - Host-to-gateway - Gateway-to-host - Host-to-host - Gateway-to-dynamic IP user - Host-to-dynamic IP user 	
<p>What type of security and system performance is required to protect the keys?</p> <ul style="list-style-type: none"> - Highest security, lowest performance - Balance security and performance - Lowest security and highest performance - Add your own definition 	
<p>What type of security and system performance is required to protect the data?</p> <ul style="list-style-type: none"> - Highest security, lowest performance - Balance security and performance - Lowest security and highest performance - Add your own definition 	

Table A-2 Key management planning

VPN Parameter	Value
Key Management Policy: Key Policy, Proposal, Transform	
Key Policy Name:	
Initiator Negotiation	
Responder Negotiation	
Key Proposal Name:	
Key Transform Name:	
Authentication Method	
Hash Algorithm	
Encryption Algorithm	
Diffie-Hellman Group	
Maximum Key Lifetime	
Maximum Size Limit	
Key Lifetime Range	
Size Limit Range	

Table A-3 Data management planning

VPN Parameter	Value
Data Management Policy: Data Policy, Proposal, AH and ESP Transform:	
Data Policy Name:	
Perfect Forward Secrecy (PFS)	
Data Proposal Name:	
AH Transform Object Name:	
AH Encapsulation Mode	
AH Authentication Algorithm	
AH Maximum Data Lifetime	
AH Maximum Size Limit	
AH Data Lifetime Range	
AH Size Limit Range	
ESP Transform Object Name:	
ESP Encapsulation Mode	
ESP Authentication Algorithm	
ESP Encryption Algorithm	
ESP Maximum Data Lifetime	
ESP Maximum Size Limit	
ESP Data Lifetime Range	
ESP Size Limit Range	

Table A-4 Network object planning

VPN Parameter	Value
Source network object name:	
IP Type	
IP Address	
Subnet Mask	
Start IP Address	
End IP Address	
Destination network object name:	
IP Type	
IP Address	
Subnet Mask	
Start IP Address	
End IP Address	

Table A-5 Connection planning

VPN Parameter	Value
Connection management: Connection, Services, Rules	
Connection Name:	
Source network object name:	
Destination network object name:	
Service Object Name:	
Rule Object Names	
Override Log Control	
Override Manual VPN Tunnel ID	
Control By Time of Day	
Control By Days	

Table A-6 Rule objects planning

VPN Parameter	Value
Rule Name:	
Action	
Protocol	
Source Port Type and Operation	
Destination Port Type and Operation	
Interface Settings	
Routing	
Direction	
Log Control	
Dynamic Tunnel Policy Name	

Table A-7 Dynamic tunnel policy planning

VPN Parameter	Value
Dynamic Tunnel Policy:	
Data Policy	
Initiation	
Connection Lifetime	

Table A-8 Authentication Information planning

VPN Parameter	Value
Authentication Information:	
Remote Key Server	
Shared Key	
Certificate Authority:	
RACDCERT Label	
Key Ring	
User ID	
Key Ring Name	

Table A-9 Dynamic VPN connection planning

VPN Parameter	Value
Dynamic VPN Connection:	
Source network object name	
Destination network object name	
Source Port	
Destination Port	
Automatic Activation	
Protocol	
Remote Key Server:	
Auth IP Type	
Auth ID	
IP address	
Host Name	
Key Server Group:	
Key Policy	
Local Key Server:	
Auth IP Type	
Auth ID	
IP address	
Host Name	
Remote Key Servers	

Table A-10 VPN On-Demand planning

VPN Parameter	Value
On-Demand name	
Source IP granularity	
Destination IP granularity	
Gateway key server	
Key server group	



Sample RACF definitions

This appendix contains samples of the RACF definitions commonly used to run a secure TCP/IP environment. We assume that you have set up the z/OS environment as described in *z/OS V1R2.0 UNIX System Services Planning*, GA22-7800. You should have a RACF group OMVSGRP and a user OMVSKERN with UID(0). These default names can be changed, if you wish, via the `/etc/options` and `/etc/profile` files. We recommend that you keep the names suggested in the planning guide for user IDs and groups; setting up a z/OS UNIX environment is non-trivial and using the same names just makes it easier without compromising security.

For a complete description of each RACF command, consult *z/OS V1R2.0 SecureWay Security Server RACF Command Language Reference*, SA22-7687.

B.1 RACF settings for UNIX System Services

- ▶ Define a superuser with a user ID of BPXROOT.

```
ADDUSER BPXROOT DFLTGRP(OMVSGRP) OMVS(UID(0) HOME('/') PROGRAM('/bin/sh'))  
NOPASSWORD
```

If in BPXPRMxx PARMLIB member SUPERUSER(BPXROOT). It is also the default.
- ▶ Activate the FACILITY class (if it is not defined).

```
SETROPTS CLASSACT(FACILITY) GENERIC(FACILITY) AUDIT(FACILITY)  
SETROPTS RACLIST(FACILITY)
```
- ▶ Create a BPX.SUPERUSER FACILITY class profile.

```
RDEFINE FACILITY BPX.SUPERUSER UACC(NONE)  
SETROPTS RACLIST(FACILITY) REFRESH
```
- ▶ Alter/add users to have OE superuser permission.

```
ALTUSER user1 OMVS(UID(7) HOME('/u/user1') PROGRAM('/bin/sh'))  
PERMIT BPX.SUPERUSER CLASS(FACILITY) ID(user1) ACCESS(READ)  
SETROPTS RACLIST(FACILITY) REFRESH
```
- ▶ Set up for a OE default user and group.

```
ADDGROUP OEGROUP OMVS(GID(17))  
ADDUSER OEDFLT NAME('USS DFLTUSER') OMVS(UID(88) HOME(/u/OEDFLT)  
RDEFINE FACILITY BPX.DEFAULT.USER APPLDATA('OEDFLT/OEGROUP')  
SETROPTS RACLIST(FACILITY) REFRESH
```

B.2 RACF settings for TCP/IP applications

- ▶ Set up the STARTED class (if it is not defined).

```
SETROPTS GENERIC(STARTED)  
SETROPTS CLASSACT(STARTED) RACLIST(STARTED)
```
- ▶ Define the TCP/IP system address space started task user ID with a UID=0.

```
ADDUSER tcpip_user DFLTGRP(OMVSGRP) OMVS(UID(0) HOME(/) PGM(/bin/sh))
```
- ▶ Assign an OMVS segment to started task user IDs specified as UID(0): (ADD or ALTUSER).

```
ADDUSER webserver_user_id DFLTGRP(OMVSGRP) OMVS(UID(0) HOME(/)  
PGM(/bin/sh))  
ADDUSER ftpd_user_id DFLTGRP(OMVSGRP) OMVS(UID(0) HOME(/) PGM(/bin/sh))  
ADDUSER SYSLOGD DFLTGRP(OMVSGRP) OMVS(UID(0) HOME(/) PGM(/bin/sh))
```
- ▶ Define started resource profiles.

```
RDEFINE STARTED TCPIP.* STDATA(USER(tcpip_user) PRIVILEGED(NO) TRUSTED(NO)  
TRACE(NO))  
RDEFINE STARTED FTPD.* STDATA(USER(tcpip_user) PRIVILEGED(NO) TRUSTED(NO)  
TRACE(NO))  
RDEFINE STARTED IMWEBSRV.* STDATA(USER(webserver_user_id) PRIVILEGED(NO)  
TRUSTED(NO) TRACE(NO))
```

Change TRACE(NO) to TRACE(YES) if you want to see a message indicating what STARTED profile was used to start the started task.
- ▶ Enable OROUTED and OMPROUTE to get started.

```
RDEFINE OPERCMDS (MVS.ROTEMGR.OMPROUTE) UACC(NONE)  
PERMIT MVS.ROTEMGR.OMPROUTE ACCESS(CONTROL) CLASS(OPERCMDS) ID(userid)  
RDEFINE OPERCMDS (MVS.ROTEMGR.OROUTED) UACC(NONE)
```

```
PERMIT MVS.ROUTEMGR.ROUTED ACCESS(CONTROL) CLASS(OPERCMD) ID(userid)
SETROPTS RACLIST(OPERCMD) REFRESH
```

B.2.1 RACF configuration for OS/390 UNIX level security

- ▶ Define the BPX.DAEMON facility class profile in RACF.

```
RDEFINE FACILITY BPX.DAEMON UACC(NONE)
```

- ▶ Allow user IDs access to the BPX.DAEMON facility class resource.

```
PERMIT BPX.DAEMON CLASS(FACILITY) ID(OMVSKERN) ACCESS(READ)
PERMIT BPX.DAEMON CLASS(FACILITY) ID(tcpip_user) ACCESS(READ)
SETROPTS RACLIST(FACILITY) REFRESH
```

- ▶ Define the BPX.SERVER facility class profile and give the Web server the UPDATE permission.

```
RDEFINE FACILITY BPX.SERVER UACC(NONE)
PERMIT BPX.SERVER CLASS(FACILITY) ID(webserver_user_id) ACCESS(UPDATE)
SETROPTS RACLIST(FACILITY) REFRESH
```

- ▶ Set up for SURROGAT class for the PUBLIC user and allow the Web server to access it.

```
SETROPTS CLASSACT(SURROGAT)
SETROPTS RACLIST(SURROGAT)
RDEFINE SURROGAT BPX.SRV.PUBLIC UACC(NONE)
PERMIT BPX.SRV.PUBLIC CLASS(SURROGAT) ID(webserver_user_id) ACCESS(READ)
SETROPTS RACLIST(SURROGAT) REFRESH
```

- ▶ Define all other BPX.* facility class profiles and permit, for example, the superuser to set APF and PROGCTL.

```
RDEFINE FACILITY BPX.DEBUG UACC(NONE)
RDEFINE FACILITY BPX.DEFAULT.USER UACC(NONE)
RDEFINE FACILITY BPX.FILEATTR.APF UACC(NONE)
RDEFINE FACILITY BPX.FILEATTR.PROGCTL UACC(NONE)
RDEFINE FACILITY BPX.SMF UACC(NONE)
RDEFINE FACILITY BPX.STOR.SWAP UACC(NONE)
RDEFINE FACILITY BPX.SUPERUSER UACC(NONE)
RDEFINE FACILITY BPX.WLMSEVER UACC(NONE)

PERMIT BPX.FILEATTR.APF CLASS(FACILITY) ID(superuser) ACCESS(READ)
PERMIT BPX.FILEATTR.PROGCTL CLASS(FACILITY) ID(superuser) ACCESS(READ)
SETROPTS RACLIST(FACILITY) REFRESH
```

- ▶ Activate program control (if not already active) to identify programs as controlled. Add DATASET profiles only if necessary.

```
SETROPTS WHEN(PROGRAM)
ADDSD 'cee.SCEERUN' UACC(READ) GENERIC
ADDSD 'websrv_h1q.SIMWMOD1' UACC(READ) GENERIC
ADDSD 'SYS1.LINKLIB' UACC(READ) GENERIC
ADDSD 'TCPIP_h1q.SEZALINK' UACC(READ) GENERIC

RDEFINE PROGRAM * ADDMEM('SYS1.LINKLIB'//NOPADCHK) UACC(READ)
RALTER PROGRAM * ADDMEM('cee.SCEERUN'//NOPADCHK) UACC(READ)
RALTER PROGRAM * ADDMEM('websrv_h1q.SIMWMOD1'//NOPADCHK) UACC(READ)
RALTER PROGRAM * ADDMEM('TCPIP_h1q.SEZALINK'//NOPADCHK) UACC(READ)
```

If you need to define a firewall, issue the following commands as well:

```
RALTER PROGRAM * ADDMEM('ICA.SICALMOD'//NOPADCHK) UACC(READ)
RALTER PROGRAM * ADDMEM('h1q.SGSKLOAD'//NOPADCHK) UACC(READ)
SETROPTS WHEN(PROGRAM) REFRESH
```

We omitted the VOLSER parameter from the ADDMEM clauses; it is no longer required.

If you code six asterisks within single quotation marks ('*****'), this refers to the current SYSRES volume, which might be useful in a sysplex environment.

B.2.2 RACF definitions to control the use of the TCP/IP operator commands

- ▶ Set up for the VARY.TCPIP command.

```
SETROPTS CLASSACT(OPERCMD) GENERIC(OPERCMD)
SETROPTS RACLIST(OPERCMD)
RDEFINE OPERCMD (MVS.VARY.TCPIP.***) UACC(NONE)
PERMIT MVS.VARY.TCPIP.***) ACCESS(CONTROL) CLASS(OPERCMD) ID(uid)
SETROPTS RACLIST(OPERCMD) REFRESH
```

B.3 Required RACF definitions to get Firewall Technologies started

- ▶ Define user and group and create the FWKERN.START.REQUEST resource profile:

```
ADDGROUP FWGRP SUPGROUP(SYS1) OMVS(GID(nnn))
ADDUSER FWKERN OMVS(HOME('/u/fw kern/')) UID(0)
      DFLTGRP(FWGRP) AUTHORITY(CREATE) UACC(ALTER) NOPASSWORD
RDEFINE FACILITY FWKERN.START.REQUEST UACC(NONE)
PERMIT FWKERN.START.REQUEST CLASS(FACILITY) ID(FWKERN) ACCESS(UPDATE)
RDEFINE STARTED FWKERN.* STDATA(USER(FWKERN))
SETROPTS RACLIST(STARTED) REFRESH
SETROPTS RACLIST(FACILITY) REFRESH
```

The HOME directory has to be created with:

```
MKDIR '/u/fw kern' MODE(7,5,5)
```

- ▶ Permit FWKERN access to the BPX.SERVER to allow ISAKMP server.

```
PERMIT BPX.SERVER CLASS(FACILITY) ID(FWKERN) ACC(READ)
```

- ▶ Grant permission to use OCSF services to FWKERN user ID:

```
PERMIT CDS.CSSM CLASS(FACILITY) ID(FWKERN) ACC(READ)
PERMIT CDS.CSSM.CRYPTO CLASS(FACILITY) ID(FWKERN) ACC(READ)
PERMIT CDS.CSSM.DATALIB CLASS(FACILITY) ID(FWKERN) ACC(READ)
SETROPTS CLASS(FACILITY) REFRESH
```

- ▶ Permit FWKERN access to start the servers and access the TCPIP data sets.

```
RDEFINE STARTED ICAPLOG.** STDATA(USER(FWKERN) GROUP(FWGRP))
RDEFINE STARTED ICAPSOCK.** STDATA(USER(FWKERN) GROUP(FWGRP))
RDEFINE STARTED ICAPPFTP.** STDATA(USER(FWKERN) GROUP(FWGRP))
RDEFINE STARTED ICAPCFG.** STDATA(USER(FWKERN) GROUP(FWGRP))
RDEFINE STARTED ICAPSTAK.** STDATA(USER(FWKERN) GROUP(FWGRP))
RDEFINE STARTED ICAPIKED.** STDATA(USER(FWKERN) GROUP(FWGRP))
PERMIT 'TCPIP.**' ID(FWKERN) ACCESS(READ)
SETROPTS RACLIST(STARTED) REFRESH
```

- ▶ Permit FWKERN to READ to BPX.SMF (for logging) and BPX.DAEMON facility class profiles.

```
PERMIT BPX.SMF CLASS(FACILITY) ID(FWKERN) ACCESS(READ)
PERMIT BPX.DAEMON CLASS(FACILITY) ID(FWKERN) ACCESS(READ)
SETROPTS RACLIST(FACILITY) REFRESH
```

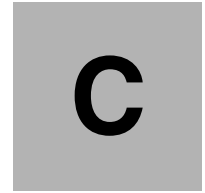

- ▶ All user IDs that will use the firewall configuration GUI need access to ICA.CFGSRV.

```
RDEFINE FACILITY ICA.CFGSRV UACC(NONE)
PERMIT ICA.CFGSRV CLASS(FACILITY) ID(userid) ACCESS(UPDATE)
SETROPTS RACLIST(FACILITY) REFRESH
```

B.4 RACF definition to manage certificate in RACF common keyring

Some required authorization to perform the basic RACDCERT actions:

```
RDEFINE FACILITY IRR.DIGTCERT.ADD UACC(NONE)
RDEFINE FACILITY IRR.DIGTCERT.ADDRING UACC(NONE)
RDEFINE FACILITY IRR.DIGTCERT.CONNECT UACC(NONE)
RDEFINE FACILITY IRR.DIGTCERT.GENCERT UACC(NONE)
RDEFINE FACILITY IRR.DIGTCERT.GENREQ UACC(NONE)
RDEFINE FACILITY IRR.DIGTCERT.LIST UACC(NONE)
RDEFINE FACILITY IRR.DIGTCERT.LISTRING UACC(NONE)
PERMIT IRR.DIGTCERT.ADD CLASS(FACILITY) ID(userid) ACC(CONTROL)
PERMIT IRR.DIGTCERT.ADDRING CLASS(FACILITY) ID(userid) ACC(UPDATE)
PERMIT IRR.DIGTCERT.CONNECT CLASS(FACILITY) ID(userid) ACC(CONTROL)
PERMIT IRR.DIGTCERT.GENCERT CLASS(FACILITY) ID(userid) ACC(CONTROL)
PERMIT IRR.DIGTCERT.GENREQ CLASS(FACILITY) ID(userid) ACC(CONTROL)
PERMIT IRR.DIGTCERT.LIST CLASS(FACILITY) ID(userid) ACC(CONTROL)
PERMIT IRR.DIGTCERT.LIST CLASS(FACILITY) ID(FWKERN) ACC(READ)
PERMIT IRR.DIGTCERT.LISTRING CLASS(FACILITY) ID(userid) ACC(UPDATE)
PERMIT IRR.DIGTCERT.LISTRING CLASS(FACILITY) ID(FWKERN) ACC(READ)
SETROPTS RACLIST(FACILITY) REFRESH
```

Default permissions for HFS files in z/OS UNIX

The following table shows the default permissions set by the system.

Use	To Create a	Default Permissions
<code>mkdir shell</code> command	Directory	owner=rwx group=rwx other=rwx In octal form: 777
MKDIR TSO command	Directory	owner=rwx group=r-x other=r-x In octal form: 755
JCL with no PATHDISP specified	Directory or file	owner=--- group=--- other=--- In octal form: 000
ISPF editor, OEDIT command, <code>oedit</code> command	File	owner=rwx group=--- other=--- In octal form: 700
vi editor	File	owner=rw- group=rw- other=rw- In octal form: 666
ed editor	File	owner=rw- group=rw- other=rw- In octal form: 666
Redirection (>)	File	owner=rw- group=rw- other=rw- In octal form: 666
<code>cp</code> command	File	Sets the output file permissions to the input file permissions.

Use	To Create a	Default Permissions
OCOPY command	File	Permission bits for a new file are specified with the ALLOCATE command, using the PATHMODE keyword, prior to entering the OCOPY command. If the PATHMODE keyword is omitted, the default is: owner=--- group=--- other=--- In octal form: 000
OPUT or OPUTX command	File	For a text file: owner=rw- group=--- other=--- In octal form: 600 For a binary file: owner=rwx group=--- other=--- In octal form: 700



Digital certificate formats supported by RACDCERT

The following information is based on OS/390 Security Server V2R10. Be sure to check if there is any updated information available.

The digital certificate must be in one of the following formats to be managed by the RACF RACDCERT command:

1. A single BER-encoded X.509 certificate.
2. A Privacy Enhanced Mail (PEM)-encoded X.509 certificate. If the input is in this format, only the Originator Certificate will be used.
3. One or more X.509 certificates contained within a PKCS #7 BER encoding. If the input is in this format, only the first certificate in the PKCS #7 encoding will be used.
4. One or more X.509 certificates contained within a PKCS #12 BER encoding.
5. A Base64-encoded certificate. The data must include this string immediately prior to the Base64 encoding:

-----BEGIN CERTIFICATE-----

The data must also include this string immediately following the Base64 encoding:

-----END CERTIFICATE-----

Note the following additional details regarding RACDCERT's certificate processing:

1. All fields as defined for X.509 Version 1 certificates must be present and must have a length greater than zero (non-null).
2. X.509 certificates with version numbers greater than 3 are not supported.
3. Version 3 certificates with critical extensions are not supported. Noncritical extensions are ignored. Critical extensions that are supported include:
 - keyUsage - { 2 5 29 37 }
 - basicConstraints - { 2 5 29 19 }
 - subjectAltName - { 2 5 29 17 }
 - issuerAltName - { 2 5 29 18 }
 - certificatePolicies - { 2 5 29 32 }

- policyMappings - { 2 5 29 33 }
 - policyConstraints - { 2 5 29 36 }
 - nameConstraints - { 2 5 29 30 }
 - extKeyUsage - { 2 5 29 37 }
 - hostIdMapping - { 1 3 18 0 2 18 1 }
4. Subject and issuer names can contain only the following string types:
 - TF8 - TAG 12 (7-bit ASCII only)
 - PRINTABLESTRING - TAG 19
 - T61STRING - TAG 20
 - IA5STRING - TAG 22
 - VISIBLESTRING - TAG 26
 - GENERALSTRING - TAG 27
 - BMPString - TAG 30 (ASCII Unicode only)
 5. For a self-signed certificate, the total length of the subject's distinguished name must be 229 characters or less. For non-self-signed certificates, the length of the subject's distinguished name is limited to 255 characters. These lengths include the X.509 identifiers (such as C= and CN=) and the dot qualifiers.
 6. If the certificate's signature is incorrect, the certificate is not added.
 7. If the certificate that is being added has the same subject's distinguished name, issuer's distinguished name, and public key as an existing certificate, and the certificate being added is not a duplicate, RACDCERT ADD processing checks the validity dates and times of the certificate. If the certificate that is being added passes the signature and date validity checks, if the end date and time on the certificate being added is later than that of the existing certificate, and if the existing certificate is not expired, it is replaced.

Care must be taken when transporting the different certificate encodings to and from an OS/390 system. Both the BER-encoded X.509 and PKCS formats are binary formats. As such, they must be transported in their exact binary format.

Note: Do not perform any ASCII-to-EBCDIC translations on these formats.

PEM and Base64, however, are text-based protocols and should be transported as text. If transporting from an ASCII system, the ASCII-to-EBCDIC translation must be performed for the PEM format and Base64 format certificates.

The following information is displayed for each digital certificate defined from the RACDCERT LIST command:

- ▶ Serial number
- ▶ Issuer's distinguished name
- ▶ Label
- ▶ Certificate ID
- ▶ Status (trusted, not trusted, or highly trusted)
- ▶ Validity dates
- ▶ Private key size
- ▶ Type of private key (ICSF or DER-encoded key), or NONE if there is no private key
- ▶ Rings
- ▶ Up to 256 bytes of the subject's name, as found in the certificate itself
- ▶ Extensions, if present (specifically, keyUsage and subjectAltName)

For more information about the certificate supported by the RACF RACDCERT command, refer to *z/OS V1R2.0 SecureWay Security Server RACF Command Language Reference*, SA22-7687.

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

IBM Redbooks

For information on ordering these publications, see “How to get IBM Redbooks” on page 448.

- ▶ *Communications Server for z/OS V1R2 TCP/IP Implementation Guide Volume 1: Base and TN3270 Configuration*, SG24-5227
- ▶ *Communications Server for z/OS V1R2 TCP/IP Implementation Guide Volume 2: UNIX Applications*, SG24-5228
- ▶ *OS/390 eNetwork Communications Server for V2R7 TCP/IP Implementation Guide Volume 3: MVS Applications*, SG24-5229
- ▶ *Communications Server for z/OS V1R2 TCP/IP Implementation Guide Volume 4: Connectivity and Routing*, SG24-6516
- ▶ *Communications Server for z/OS V1R2 TCP/IP Implementation Guide Volume 5: Availability, Scalability, and Performance*, SG24-6517
- ▶ *Communications Server for z/OS V1R2 TCP/IP Implementation Guide Volume 6: Policy and Network Management*, SG24-6839
- ▶ *TCP/IP in a Sysplex*, SG24-5235
- ▶ *Managing OS/390 TCP/IP with SNMP*, SG24-5866
- ▶ *Secure e-business in TCP/IP Networks on OS/390 and z/OS*, SG24-5383
- ▶ *TCP/IP Tutorial and Technical Overview*, GG24-3376
- ▶ *Migrating Subarea Networks to an IP Infrastructure Using Enterprise Extender*, SG24-5957
- ▶ *Networking with z/OS and Cisco Routers: An Interoperability Guide*, SG24-6297

Other resources

These publications are also relevant as further information sources:

- ▶ *z/OS V1R2.0 UNIX System Services Planning*, GA22-7800
- ▶ *z/OS V1R2.0 UNIX System Services User's Guide*, GA22-7801
- ▶ *z/OS V1R1.0-V1R2.0 MVS Initialization and Tuning Guide*, SA22-7591
- ▶ *z/OS V1R2.0 C/C++ Programming Guide*, SC09-4765
- ▶ *z/OS V1R2.0 C/C++ Run-Time Library Reference*, SA22-7821
- ▶ *z/OS V1R2.0 CS: IP Migration*, GC31-8773
- ▶ *z/OS V1R2.0 CS: IP Configuration Guide*, SC31-8775
- ▶ *z/OS V1R2.0 CS: IP Configuration Reference*, SC31-8776
- ▶ *z/OS V1R2.0 CS: IP User's Guide and Commands*, SC31-8780
- ▶ *z/OS V1R2.0 CS: IP System Administrator's Commands*, SC31-8781

- ▶ *z/OS V1R2.0 CS: IP Diagnosis*, GC31-8782
- ▶ *z/OS V1R2.0 CS: IP Messages Volume 1 (EZA)*, SC31-8783
- ▶ *z/OS V1R2.0 CS: IP Messages Volume 2 (EZB)*, SC31-8784
- ▶ *z/OS V1R2.0 CS: IP Messages Volume 3 (EZY)*, SC31-8785
- ▶ *z/OS V1R2.0 CS: IP Messages Volume 4 (EZZ-SNM)*, SC31-8786
- ▶ *z/OS V1R2.0 CS: IP Application Programming Interface Guide*, SC31-8788

Referenced Web sites

These Web sites are also relevant as further information sources:

- ▶ The z/OS Web pages
<http://www-1.ibm.com/servers/eserver/zseries/zos/installation/installz12.html>

How to get IBM Redbooks

You can order hardcopy Redbooks, as well as view, download, or search for Redbooks at the following Web site:

ibm.com/redbooks

You can also download additional materials (code samples or diskette/CD-ROM images) from that site.

IBM Redbooks collections

Redbooks are also available on CD-ROMs. Click the CD-ROMs button on the Redbooks Web site for information about all the CD-ROMs offered, as well as updates and formats.

Index

Symbols

/etc/hosts.equiv 346
/etc/services 420
/etc/syslog.conf 419
_BPXK_SETIBMOPT_TRANSPORT 65

A

access control directives 371
access control list (ACL) 30
ACL 30
AES 11
AF_INET socket 420
AF_UNIX socket 420
aggressive mode 94
AH 92
ALTUSER command 200
Apache 399
APAR
 PQ41777 372
application level gateway 88
Authentication Header (AH) 95
authentication server (AS) 192
authenticator 194

B

basicConstraints 445
BIND 60
BPX.DAEMON 32, 48
BPX.DEBUG 44–45, 49
BPX.DEFAULT.USER 42
BPX.FILEATTR.APF 44, 49
BPX.FILEATTR.PROGCTL 44, 49
BPX.FILEATTR.SHARELIB 44
BPX.SERVER 32, 44, 49, 369, 372
BPX.SMF 49
BPX.SRV 369
BPX.STOR.SWAP 50
BPX.SUPERUSER 44, 47
BPX.WLMSEVER 50
BPXROOT 48

C

CA 14, 17, 383
CDMF 10
Certificate Authority (CA) 14, 188, 383
certificate management
 RACDCERT 205
 RACF common keyring 207
 SSL 235–236
certificate name filtering 393
Certificate Version Number 205
certificatePolicies 445

chmod command 38
CHOWN.UNRESTRICTED 44
Commercial Data Masking Facility (CDMF) 10
Common INET (C-INET) 60
Common Name 206
configuration client 205
Country 206
Cryptographic Coprocessor Feature 379

D

Data Encryption Standard (DES) 10
DCAS
 RACDCERT 205
default user 42
default_realm statement 201
DELUSER command 200
demilitarized zone (DMZ) 89
denial of service (DoS) 198
DES 10, 207
Diffie-Hellman 13, 94
digital certificate 15, 402, 445
 Certificate Authority (CA) 17
 management in OS/390 and z/OS 17, 205
 Public Key Infrastructure (PKI) 15
 security considerations 16
digital certificates in z/OS 205
digital signatures 20
distinguished name (DN) 15, 205, 404
DMZ 89
DN 15, 205, 404, 409
Domino Go Webserver 394
DoS 198

E

elliptic curve 13
Encapsulating Security Payload (ESP) 95
encryption algorithms
 AES 11
 CDMF 10
 DES 10
 Diffie-Hellman 13
 elliptic curves 13
 IDEA 11
 performance issues 17
 RC2 10
 RC4 11
 triple DES 10
environment variable
 SSL_KEYRING 408
ESP 92
extended attributes 44
extKeyUsage 446
extranet 86
EZB.NETACCESS 67

EZB.NETSTAT 79
EZB.PORTACCESS 71
EZB.STACKACCESS 61

F

Fast Response Cache Accelerator (FRCA) 375
file security packet 37
File Transfer Protocol (FTP) 256
firewall 85, 99

- application level gateway 88
- categories 87
- general guidelines 86
- NAT 89
- packet filtering 87
- proxy 89
- SOCKS 89
- VPN 89

Firewall Technologies 440

- CFGSRV 106
- configuration client
 - gskkyman 205
 - setup 107
- configuration panel 108–109
- Configuration Server and Client 105
- Configuration Server setup 106
- configuring 103
- RACF definitions 102
- sample configuration files 102
- simple scenario 105
- SSL configuration 105
- tunnel export file 118

FRCA 375
FTP 89
FTP proxy 89

G

generic servers 63
GID 30

- effective 40
- real 40
- saved 40

gskkyman 105, 205, 380, 405

- certificate request file 238
- key pair file 238
- stash file 238

GSSAPI protocol 198

H

HFS 36, 443

- access permissions 36
- permission bits 30, 294, 443

HMAC 20
hostIdMapping 446
HTTP 89
HTTP Server 367, 402

- %%CERTIF%% 372
- %%CLIENT%% 372
- %%LDAP%% 396

%%SAF%% 378
%%SERVER%% 372
/etc/httpd.conf 369, 381
access control directives 371
basic authentication 370
Certificate Authority (CA) 383
certificate management 235–236
certificate name filtering 393
DefProt directive 400
digest authentication 371
DIGTCERT class 387
exec directive 375
Fast Response Cache Accelerator (FRCA) 375
Global Server IDs 379
GroupFile subdirective 396
gskkyman 205
htadm command 396
LDAPInfo directive 395
mask subdirective 376
pass directive 375
PasswdFile subdirective 376, 378, 396
PrimaryLdapServer 397
Protect directive 377
protection directive 375, 377
RACDCERT 205, 387, 392, 400
RACF auto registration 388, 393
SAF control 369
search filter 396
SecondaryLdapServer 397
security functions 368
server ID subdirective 376
SSL 378

- client authentication 382
- scenarios 380
- server authentication 380
- server certificate 381
- Version 2 378
- Version 3 379, 382

SSLClientAuth directive 384
SURROGAT class 369
surrogate user ID 372, 400
threads 369
user ID directive 377
UserNameFilter subdirective 399
UserSearchBase subdirective 399
-vv trace 385
X.509 digital certificate 373, 387

I

ICSF 379
ICSS 378
IDEA 11
identity protection 94
IKE 93, 205

- RACDCERT 205

InetD 54, 344
International Data Encryption Algorithm (IDEA) 11
Internet 86
Internet Engineering Task Force (IETF) 186
intranet 86

- IP
 - filtering 89
 - IPSec 91–92
 - aggressive mode 94
 - AH 92
 - authentication 93
 - ESP 92
 - identity protection 94
 - IKE 93
 - ISAKMP 93
 - main mode 94
 - Security Association (SA) 93
 - SPI 93
 - transmitting data 95
 - transport mode 95
 - tunnel mode 95–96
 - ISAKMP 93
 - issuerAltName 445
- K**
- KDC 199
 - KDC statement 201
 - Kerberos 192, 345, 349
 - assumptions 198
 - authentication server (AS) 192
 - authenticator 194
 - denial-of-service (DoS) 198
 - implementation in z/OS 198
 - inter-realm operation 197
 - Key Distribution Center (KDC) 192
 - principal identifier 192
 - realm 192
 - ticket granting ticket (TGT) 192
 - Version 5 192
 - KERBLINK class 200
 - Key Distribution Center (KDC) 192
 - keyring 203, 300
 - keyUsage 445
 - kpasswd_server statement 201
- L**
- LDAP 394, 402
 - /etc/ldap/slapd.conf 407
 - access control 403
 - ACL 403
 - anonymous authentication 404
 - authentication 403
 - basic authentication 404
 - confidentiality 403
 - CRAM-MD5 404
 - EXTERNAL 404, 408–409
 - gskkyman 205, 405
 - key database 406
 - HTTP Server 399
 - integrity 403
 - ldapadd 408
 - ldapcp 408
 - ldapdelete 408
 - ldapmodify 408
 - ldapmodrdn 408
 - LDIF 398
 - ldif2db 398
 - RACDCERT 205
 - RACF
 - RACF-style distinguished name 404
 - RFC 2222 404
 - SASL 403, 408–409
 - security option 407
 - security overview 403
 - security policy 404
 - serverAuth 407
 - Simple Authentication and Security Layer 404
 - SSL 403
 - CA 405
 - self-signed certificate 405
 - SSL_KEYRING environment variable 408
 - sslAuth 408
 - sslCipherSpecs 407
 - sslKeyRingFile 407
 - sslKeyRingFilePW 407
 - sslKeyRingPWStashFile 407
 - userPassword attribute 399
 - versions 404
 - WebSphere Application Server (WAS) 399
 - X.509 certificate 404
 - Lightweight Directory Access Protocol (LDAP) 402
 - Locality 206
- M**
- main mode 94
 - MD2 19
 - MD5 19
 - message authentication 17
 - message authentication code (MAC) 7, 188
 - HMAC 20
 - message digest algorithms
 - MD2 19
 - MD5 19
 - SHA-1 19
 - mutual authentication 197
- N**
- nameConstraints 446
 - NETSTAT 79
 - Network Address Translation (NAT) 89
 - Network File System (NFS) 288
 - NFS 288
 - /etc/exports 289
 - df command 296
 - EXPORTS data set 289–290
 - EXPORTS security 294, 296
 - mount command 293
 - mvslogin command 292, 296
 - mvslogout command 292
 - PC-NFS 290, 296
 - qmount command 296
 - SAF checking 289
 - SAF security 294, 296

- SAFEXP security 297
- security parameter 289
- showmount command 290

nonce 194

O

- Oakley 93
- OMPROUTE 55
- onetstat 79
- OPERCMD5 76
- Organization 206
- Organizational-unit 206
- Organization-name 206
- ORouteD 55
- OS/390 Firewall Technologies 89, 99

P

- packet filtering 87
- password-guessing attacks 198
- permission bits 30, 36, 294, 443
- PKCS#12 format 389, 445
- PKCS#7 format 389, 445
- PKDS 207
- PKI 14
- policy agent
 - gskkyman 205
 - RACDCERT 205
 - security 362
 - SSL support 363
- policyConstraints 446
- policyMappings 446
- principal identifier 192
- program control 50
- protection directives 375
- proxies 88
- proxy 89
- Public Key Data Set (PKDS) 207
- Public Key Infrastructure (PKI) 14
 - digital certificate 15
- public/private key encryption 11

Q

- Quality of Service (QoS) 402
 - LDAP_SSL statement 364
 - policy agent
 - security 362
 - SSL 363
 - policy agent with SSL connections 363
 - SSL 363

R

- RACDCERT RACF command 205, 387, 392, 445
- RACF 25, 100
 - auto registration 388, 393
 - BPX.DAEMON 32, 48
 - BPX.DEBUG 44–45, 49
 - BPX.DEFAULT.USER 42
 - BPX.FILEATTR 44

- BPX.FILEATTR.APF 44, 49
- BPX.FILEATTR.PROGCTL 49
- BPX.FILEATTR.SHARELIB 44, 49
- BPX.SERVER 32, 44, 49, 369, 372
- BPX.SMF 49
- BPX.SRV 369
- BPX.STOR.SWAP 50
- BPX.SUPERUSER 44, 47
- BPX.WLMSEVER 50
- certificate management 203
- certificate name filtering 393
- common keyring 234, 441
- DIGTCERT 387
- OMVS address space 52
- OMVS segment 43, 294, 344
- program control 50
- RACDCERT 205, 387, 392, 445
- RACROUTE 26
- sample definitions 437
 - Firewall Technologies 440
 - OS/390 UNIX level security 439
 - RACF common keyring 441
 - TCP/IP applications 438
 - TCP/IP operator commands 440
 - UNIX System Services 438
- STARTED 46
- started task 46
- SURROGAT class 369, 376
- UNIX System Services 28
- UNIXPRIV class 44
 - CHOWN.UNRESTRICTED 44
 - SUPERUSER.FILESYS 45
 - SUPERUSER.IPC.RMID 45
 - SUPERUSER.PROCESS 45
 - SUPERUSER.SETPRIORITY 45
- RACF common keyring 207, 441
- RACLIST 73
- RC2 10
- RC4 11
- realm 192
- Redbooks Web site 448
 - Contact us xiii
- Request for Comments (RFC)
 - RFC 1510 192
 - RFC 2222 404
 - RFC 2253 15
 - RFC 2412 93
 - RFC 2941 345
 - RFC 2942 345
 - RFC 2946 345
 - RFC 2952 345
 - RFC 2953 345
- RESTRICTLOWPORTS 71
- rexecd 346
- rlogind 346
- routing daemons 55
- routing protocol security 354
- RSA 13
- rshd 346
 - Kerberos support 349

S

- SA 93
- Secure Hash Standard (SHS) 19
- Security Association (SA) 93
- Security Parameter Index (SPI) 93
- Security Server 25
- self-signed certificate 191, 203–204
- SERVAUTH class 61
- SHA-1 19
- Simple Network Management Protocol (SNMP) 358
- SNMP 358
 - community-based security 359
 - MIB 358
 - NetView SNMP 359
 - osnmp 359
 - PDU 359
 - SET commands 361
 - SNMP agent 358
 - SNMP subagent 358
 - SNMPv1 359
 - SNMPv2c 359
 - SNMPv2u 359
 - SNMPv3 359
 - User-Based Security Model 360
 - USM 360
 - VACM 360
 - View-Based Access Control Model 360
- SOCKS 89
- SOCKS server 89
- SSL 205, 378, 403
 - Certificate Authority (CA) 203, 383
 - client authentication 187, 382
 - DES 378
 - MAC 188
 - MD5 379
 - message authentication code (MAC) 188
 - public-private key pair 236
 - RC2 378
 - RC4 378
 - record protocol 188
 - self-signed certificate 191, 237
 - server authentication 241
 - server certificate 241, 381
 - SHA 379
 - symmetric encryption keys 188
 - Triple DES 378
 - Version 3 379, 382
- SSL-enabled applications 186
- STARTED RACF class 46
- started task user ID 46
- stash file 300
- State-or-province 206
- sticky bit 51
- subjectAltname 445
- superuser 43
- SUPERUSER.FILESYS 45
- SUPERUSER.IPC.RMID 45
- SUPERUSER.PROCESS 45
- SUPERUSER.SETPRIORITY 45
- syslogd

- /etc/services 420
- /etc/syslog.conf 419
- AF_INET socket 420
- AF_UNIX socket 420
- isolation 420

T

- TCPCONFIG 71
- Telnet server 344
 - Kerberos support 345, 349
- TFTP 287
- ticket granting ticket (TGT) 192
- ticket-granting server (TGS) 193
- Title 206
- TLS 191, 209
- TN3270 208
 - gskkyman 205
 - RACDCERT 205
- transport mode 95
- Triple-DES 10, 207
- TSO ISHELL 40
- tunnel definition 109
- tunnel export file 118
- tunnel mode 96

U

- UID 30, 289
 - effective 40
 - real 40
 - saved 40
- Universal Access (UACC) 61
- UNIX level security 32, 439
- UNIX rexecd 346
- UNIX rlogind 346
- UNIX rshd 346
 - Kerberos support 349
- UNIX System Services 25, 438
 - access control list (ACL) 30
 - BPX.DAEMON 32
 - BPX.SERVER 32
 - BPXROOT 48
 - default user 42
 - defining users 40
 - extended attributes 44
 - GID 293
 - group ID 30
 - HFS permission bits 30, 36, 294, 443
 - home directory 42
 - ISPF shell 38
 - program control 50
 - security 25
 - sticky bit 51
 - superuser 43
 - superuser granularity 44
 - UID 293
 - UNIX level security 32
 - user ID 30
- UNIX Telnet server 344
 - Kerberos support 345, 349

UNIXPRIV RACF class 44
SUPERUSER.IPC.RMID 45
SUPERUSER.PROCESS 45
SUPERUSER.SETPRIORITY 45

V

VeriSign 188
virtual private network (VPN) 89, 91
 planning worksheets 425
VPN 89, 91

W

Web server 367
WebSphere Application Server 399

X

X.509 digital certificate 402, 445



Redbooks

Communications Server for z/OS V1R2 TCP/IP Implementation Guide Volume 7: Security

(1.0" spine)

0.875" x 1.498"

460 <-> 788 pages



Communications Server for z/OS V1R2 TCP/IP Implementation Guide

Volume 7: Security



Introduces security concepts including cryptography and digital certificates

Details z/OS application security using SSL, TLS, and Kerberos

Covers SAF-based and network security

The Internet and enterprise-based networks have led to a rapidly increasing reliance upon TCP/IP implementations. The zSeries platform provides an environment upon which critical business applications flourish. The demands placed on these systems is ever-increasing and such demands require a solid, scalable, highly available, and highly performing operating system and TCP/IP component. z/OS and Communications Server for z/OS provide for such a requirement with a TCP/IP stack that is robust and rich in functionality. The *Communications Server for z/OS TCP/IP Implementation Guide* series provides a comprehensive, in-depth survey of CS for z/OS.

Volume 7 covers z/OS TCP/IP security issues. First, we begin with a survey of security issues and available methodologies for handling these issues such as cryptography. We discuss SAF-based authorization techniques available on z/OS. We continue by presenting topics related to network security such as Firewall, VPN, and IPsec. Finally, we provide an in-depth look into application security including topics such as SSL, TLS, and Kerberos. We frame our discussion within application-specific chapters that cover everything from Telnet to FTP to DNS. Because of the varied scope of CS for z/OS, this volume is not intended to cover all aspects of it. The main goal of this volume is to provide an insight into the different functionality available for securing the CS for z/OS environment. For more information, including applications available with CS for z/OS IP, please reference the other volumes in the series.

**INTERNATIONAL
TECHNICAL
SUPPORT
ORGANIZATION**

**BUILDING TECHNICAL
INFORMATION BASED ON
PRACTICAL EXPERIENCE**

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

**For more information:
ibm.com/redbooks**

SG24-6840-00

ISBN 0738426385