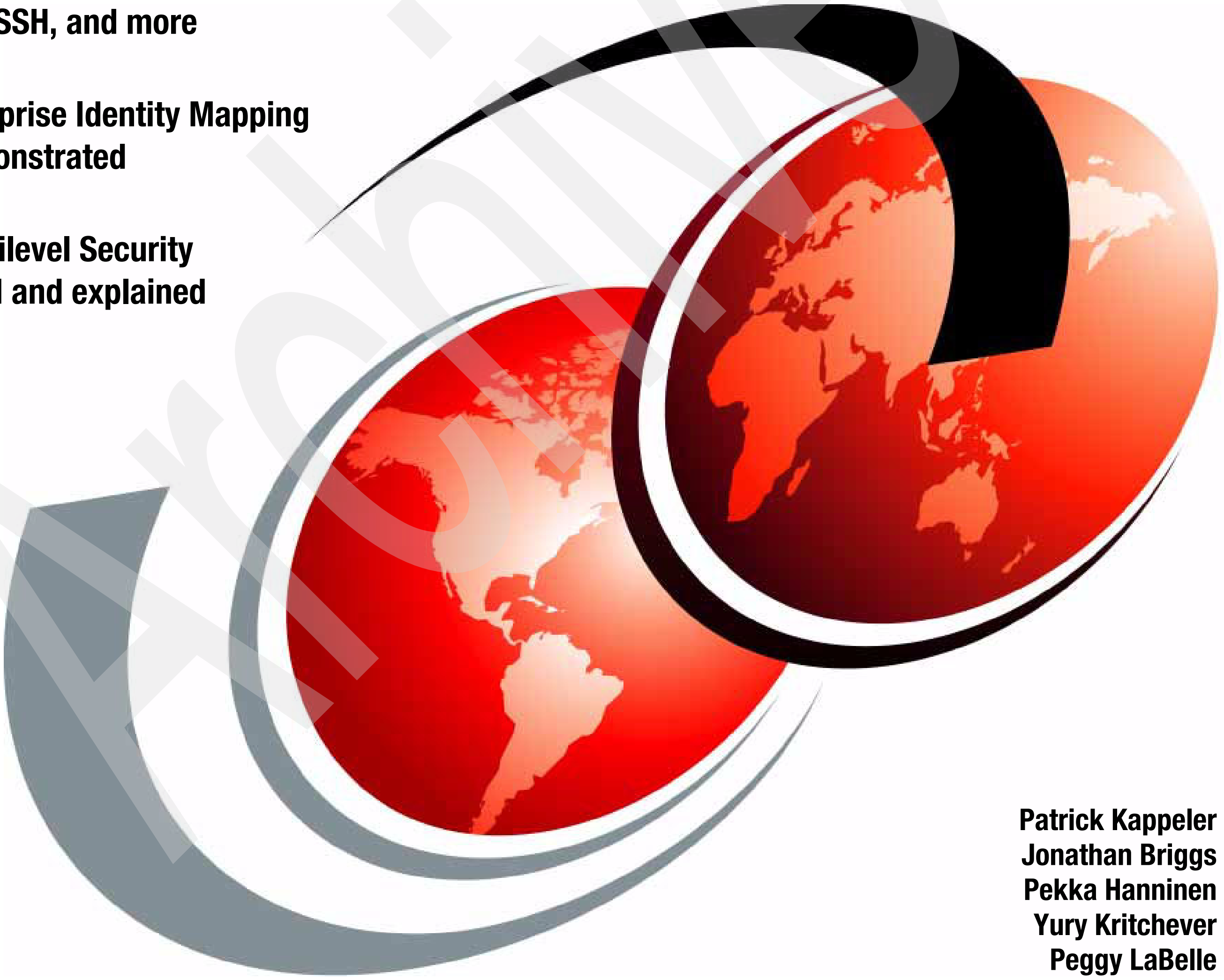
IBM

z/OS 1.6 Security Services Update

The latest on z/OS LDAP, RACF, System SSL, OpenSSH, and more

z/OS Enterprise Identity Mapping (EIM) demonstrated

RACF Multilevel Security introduced and explained



Patrick Kappeler
Jonathan Briggs
Pekka Hanninen
Yury Kritchever
Peggy LaBelle

Redbooks

ibm.com/redbooks

International Technical Support Organization

z/OS 1.6 Security Services Update

July 2005

SG24-6448-00

Note: Before using this information and the product it supports, read the information in "Notices" on page xv.

First Edition (July 2005)

This edition applies to Version 1, Release 6 of z/OS (product number 5694-A01).

Contents

Figures

Notices

Trademarks

Preface

This IBM Redbook describes the z/OS® Security services provided by z/OS as of z/OS 1.6.

As this is a vast subject, some of these services have already been addressed by other redbooks, usually in the OS/390®-z/OS 1.2 time frame. We are therefore concentrating in this book on new services or services that have gone under a noticeable evolution since they were described in previous redbooks; and we are providing, whenever appropriate, simple examples of utilization.

The first chapter is a summary of all Security services available in z/OS 1.6 today with pointers to either chapters in this book or to other redbooks for additional information.

This publication addresses the enhancements in RACF® Security services in the domains of the Unix Security Services, the digital certificate handlings, and the more traditional PADS (Program Access to d Data Sets) function, which have all been enhanced to better match the eBusiness needs in terms of compliance with standards and improved Security. Note that RACF MLS (Multilevel Security) is also introduced in this book, with a practical example of its application to TCP/IP Security.

We also provide the latest updates on z/OS LDAP server and client functions, and explain how the new RACF Password Enveloping function can be used to securely propagate new RACF passwords to other registries via the LDAP protocol.

The IBM @server™ Enterprise Identity Mapping function is explained, and a demo running on z/OS is also described in detail in this book.

The Network Authentication Service (Kerberos) has been enhanced along the z/OS releases since its initial release at OS/390 2.10. We provide a description of these enhancements and we are explaining how to run the new C programs, which are now delivered in z/OS to test the setup of Network Authentication Services.

System SSL and the gskkyman UNIX® utility enhancements are described and explained. The TLS support, the new cipherspecs, and the now available PKCS#7 functions are also explained.

This book introduces the z/OS OpenSSH component, which now makes available on z/OS the very popular UNIX OpenSSH suite of secure connectivity tools. Examples of use with a PuTTY SSH client are also provided.

The team that wrote this redbook

This redbook was produced by a team of specialists from around the world working in the EMEA Products and Solutions Support Center (PSSC) in Montpellier (France,) on behalf of the International Technical Support Organization, Poughkeepsie Center.

Patrick Kappeler led this redbook project. He joined IBM® in 1970 as a diagnostic programs designer. He has held several specialist and management positions, as well as international assignments, all dealing with S/390® and zSeries® technical support. He has ben part of the EMEA Products and Solutions Support Center, located in Montpellier (France) since 1996, where his domain of expertise is the On Demand Business Security on zSeries. He extensively writes and presents on this topic.

Jonathan Briggs is an IT Specialist working on the OS/390 Strategic Outsourcing team based in Leeds, UK. He is a systems programmer and his main skills are IT Security. He has 20 years of experience in the IT industry. His areas of expertise include IT Security on mainframe platforms, NT, and third-party security products. He also has experience in systems programming and he participated in writing the redbooks *Putting the Latest Security Features to Work*, SG24-6540, and *z/OS WebSphere® and J2EE Security Handbook*, SG24-6846.

Pekka Hanninen is an IT Specialist working on the Integrated Technology Services team in Finland. He has 30 years of experience in IBM Large Systems software. He has worked at IBM for eight years. His areas of expertise include RACF, cryptography, and security administration. He holds certificates for CISSP, CISA, and CISM.

Yury Kritchever is an Adviser IT Specialist in z/OS MVS™ Support, IBM Global Services Canada, located in Markham, Canada. He has 26 years of experience in S/390 and z/Series MVS system programming and support, and holds an M.S. degree in Computer Science from Moscow Institute of Management. His areas of expertise include OS/390 and z/Series MVS, RACF, TSO/E, HCD, as well as Parallel Sysplex® customization and support.

Peggy LaBelle is an Advisory Software Engineer with IBM. Her career has been spent as a designer and developer of enhancements to the MVS, OS/390, and z/OS products developed at IBM Poughkeepsie. She is currently responsible for the design of the z/OS implementation of Enterprise Identity Mapping. She has presented numerous topics at security conferences, most recently at the 2004 Networking and Security Conference sponsored by IBM.

Many thanks to the following people for their highly appreciated direct or indirect contribution to this project:

Chris Rayns
Ela Buslovich
Julie Czubik
International Technical Support Organization, Poughkeepsie Center

Noel Richard
Jean-Jacques Noguera
Alain Dalmier
EMEA Products and Solutions Support Center, Montpellier (France)

Roland Grant
Matthew Yates
Norbert Verbestel
IBM Global Services

Joseph Harfouch
Contractor (IBM Australia)

Mike Holmans
British Telecommunications plc

Jun Ogata
Walt Farrell
Eric Rosenfeld
John Willy Wilson
Carolyn McLaughlin
Laurie Ward
Bruce Wells

Russell D Hardgrove
z/OS Security Server Development and System Support, Poughkeepsie, New York

Ronald Hoffman
Alyson Comer
Jay Brodfuehrer
Kenneth Morgan
Kim Worm
Jonathan Cottrell
Matthew J Robins
z/OS Kerberos, SSL and LDAP Development, Poughkeepsie, New York

Wai Choi
James Sweeny
z/OS cryptography and PKI Development, Poughkeepsie, New York

Erin Farr
z/OS Unix System Services Development, Poughkeepsie, New York

Become a published author

Join us for a two- to six-week residency program! Help write an IBM Redbook dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You'll team with IBM technical professionals, Business Partners or clients.

Your efforts will help increase product acceptance and client satisfaction. As a bonus, you'll develop a network of contacts in IBM development labs, and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

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 email to:

  redbook@us.ibm.com

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. HYJ Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

1

Overview of z/OS Security Services

In this chapter we briefly describe the major Security features that became part of z/OS along with past releases of OS/390 and z/OS. Whenever appropriate we also provide references to previous redbooks that more specifically address these topics.

1.1 Packaging of the imbedded Security functions at z/OS 1.6

The Security function packaging as delivered in z/OS 1.6 is composed of the following services. Note that only the z/OS Security Server, now containing the RACF product alone, is the only z/OS component that requires a licence to be used.

1.1.1 z/OS Cryptographic Services

Here we review the z/OS Cyrpotgraphic Services.

ICSF

The Integrated Cryptographic Service Facility is the z/OS component that both drives the zSeries hardware cryptographic coprocessors and provides the IBM Common Cryptography Architecture (CCA) API for the applications or middlewares to invoke the hardware cryptographic services. ICSF is integrated in the base z/OS since OS/390 2.4 is upgraded to support new coprocessors and cryptographic functions.

The following redbooks specifically address the hardware coprocessors and ICSF support as provided on the different S/390 and zSeries models:

- *Exploiting S/390 Hardware Cryptography with Trusted Key Entry*, SG24-5455
- *S/390 Crypto PCI Implementation Guide*, SG24-5942
- *zSeries Crypto Guide Update*, SG24-687
- *IBM @server zSeries 990 (z990) Cryptography Implementation*, SG24-7070

The z/OS reference books pertaining to ICSF are:

- *z/OS ICSF Overview*, SA22-7519
- *z/OS ICSF System Programmer's Guide*, SA22-7520
- *z/OS ICSF Application Programmer's Guide*, SA22-7522
- *z/OS ICSF Administrator's Guide”* SA22-7521
- *z/OS ICSF Messages*, SA22-7523
- *z/OS Trusted Key Entry Workstation User's Guide 2000*, SA22-7524

OCSF

The Open Cryptographic Service Facility is the OS/390 implementation of the Intel® Common Data Security Architecture (CDSA) API. OCSF is part of the base z/OS since OS/390 2.7 and the last update to the product occurred at OS/390 2.10 - z/OS 1.1. It is described in the redbooks *OS/390 Security Server 1999 Updates Technical Presentation Guide*, SG24-5627, and *OS/390 Security Server 1999 Updates Installation and Implementation Guide*, SG24-5629.

The z/OS reference books pertaining to OCSF are:

- *z/OS Open Cryptographic Services Facility Application Programming*, SC24-5899
- *z/OS Open Cryptographic Services Facility Service Provider Module Developer's Guide and Reference*, SC24-5900

System SSL

System SSL is a base component of z/OS introduced at OS/390 2.7. It is a generic set of API applications that can use to protect TCP/IP socket communications using the Secure Socket Layer (SSL), and now the new Transaction Layer Security (TLS), protocol. System SSL is steadily updated to extend the protocol capabilities as proposed in existing or new RFCs and to take advantage of new hardware cryptographic coprocessors technology.

The z/OS reference book pertaining to System SSL is *z/OS Cryptographic Services System Secure Sockets Layer Programming*, SC24-5901.

System SSL is addressed in this book in Chapter 9, “z/OS System SSL” on page 207.

z/OS PKI Services

The z/OS Public Key Infrastructure (PKI) Services is introduced at z/OS 1.3 as a base component of z/OS. It is an industry-class Registration and Certification Authority software that runs on z/OS, exploiting the capability of RACF, or an equivalent product, to protect the Certification Authority RSA private key and using, whenever available, the hardware cryptography services. The z/OS PKI Services are specifically addressed in the redbook *Implementing PKI Services on z/OS*, SG24-6968.

The z/OS reference book pertaining to the z/OS PKI Services is *z/OS Cryptographic Services PKI Services Guide and Reference*, SA22-7693.

1.1.2 z/OS Security Server

The z/OS Security Server is an optional feature that used to contain several products before its content was revisited at z/OS 1.5. Prior to this z/OS release it contained:

- RACF
- The OS/390 Firewall Technologies FTP Proxy and SOCKS servers
- The OS/390 Firewall technologies IPSec key server
- The LDAP Directory Server
- The Network Authentication Service
- the Open Cryptographic Enhanced Plug-in (OCEP)
- The DCE Security Server

Starting with z/OS 1.5, the z/OS Security Server only contains the Resource Access Control Facility (RACF) product. Note that although RACF is always delivered with z/OS, it takes having proper licensing to be authorized to use it.

Some of the z/OS reference books pertaining to RACF are:

- *z/OS Security Server RACF Security Administrator's Guide*, SA22-7683
- *z/OS security Server RACF Command Language Referenc*e, SA22-7687
- *z/OS Security Server RACF Auditor's Guide*, SA22-7684
- *z/OS Security Server RACF Callable Services*, SA22-7691
- *z/OS Security Server RACF Macros and Interfaces*, SA22-7682
- *z/OS Security Server RACF System Programmers Guide*, SA22-7681

The RACF improvements are addressed in Chapter 2, “RACF Security Server enhancements” on page 7; Chapter 3, “Multilevel Security and RACF” on page 61; an Chapter 4, “MLS as applied to TCP/IP communications” on page 83.

1.1.3 z/OS Integrated Security Services

In this section we review z/OS Integrated Security Services.

Firewall technologies

These z/OS integrated firewall protection mechanisms are available in the base z/OS, beginning with OS/390 2.5, and are specifically addressed by the following redbooks:

- *Stay Cool on OS/390: Installing Firewall Technology*, SG24-2046
- *Implementing VPNs in a z/OS Environment*, SG24-6530

The z/OS reference book pertaining to firewall technologies is *z/OS Security Server Firewall Technologies*, SC24-5922.

LDAP Directory Server

The LDAP Directory Server has been introduced at OS/390 2.5 as a licensed product included in the Security Server. The requirement for a license was dropped at OS/390 2.8.

New functions of the z/OS LDAP Directory Server are described in Chapter 5, "z/OS Integrated Security Services LDAP" on page 105; and in Chapter 6, "RACF Password Enveloping and z/OS LDAP Change Log" on page 149, in this book. More basic information can be found the following redbooks:

- *OS/390 Security Server 1999 Updates Technical Presentation Guide*, SG24-5627
- *OS/390 Security Server 1999 Updates Installation and Implementation Guide*, SG24-5629
- *Putting the Latest z/OS Security Features to Work*, SG24-6540

The z/OS reference books pertaining to the LDAP server and client are:

- *z/OS Integrated Security Services LDAP Server Administration and Use*, SC24-5923
- *z/OS Integrated Security Services LDAP Client Programming*, SC24-5924

The z/OS LDAP improvements are addressed in Chapter 5, "z/OS Integrated Security Services LDAP" on page 105.

DCE Security Server

The Distributed Computing Environment (DCE) Server has been stabilized at OS/390 2.5. It is addressed in:

- *RACF Support for Open Systems Technical Presentation*, GG26-2005
- *MVS/ESA™ OpenEdition DCE: RACF and DCE Security Interoperation*, GG24-2526

OCEP

The Open Cryptographic Enhanced Plug-In (OCEP) allows RACF, or an equivalent product, to act as a Trust Policy and Data Library for OCSF. OCEP is described, along with OCSF, in the redbooks *OS/390 Security Server 1999 Updates Technical Presentation Guide*, SG24-5627, and *OS/390 Security Server 1999 Updates Installation and Implementation Guide*, SG24-5629.

The z/OS reference book pertaining to OCEP is *z/OS SecureWay® Security Server Open Cryptographic Enhanced Plug-ins Application Programming*, SC24-5925.

Network Authentication Service

Network Authentication Service is the z/OS implementation of the Kerberos authentication protocol and Key Distribution Center. It is in the base z/OS since OS/390 2.10, retrofitted via PTF to OS/390 2.8. Network Authentication Service is addressed in Chapter 8, "z/OS Network Authentication Service (Kerberos)" on page 187. More basic information is available in the redbook *Putting the Latest z/OS Security Features to Work*, SG24-6540.

EIM

Enterprise Identity Mapping (EIM) is a set of APIs that provide for applications to easily get from an installation central repository, mapping information of an installation-wide user identity to a local platform identity. This local identity is intended to be eventually used by the local, unaltered, access control mechanism. EIM is available on z/OS since z/OS 1.4 and is addressed in Chapter 7, "z/OS Enterprise Identity Mapping (EIM) in a nutshell" on page 159.

The EIM reference book is *z/OS Integrated Security Services Enterprise Identity Mapping (EIM) Guide and Reference*, SA22-7875.

1.1.4 Additional products

There are currently two additional products, not delivered in z/OS but orderable at no cost.

PDAS

The Policy Director Authorization Services (PDAS), Program Number 5655-F95, have been introduced at z/OS 1.3, with a retrofit to OS/390 2.10 by SPE. It provides applications with the capability of interfacing with a Tivoli® Access Manager for eBusiness authorization manager (formerly called *Policy Director*, hence the PDAS name). the PDAS services are a subset of the aznAPI, which allows applications to get user credentials and user authorization data from Tivoli Access Manager. As of the writing of this book, the only known IBM product exploiting this facility in z/OS is WebSphere MQ.

The PDAS reference book is *Policy Director Authorization Services for z/OS and OS/390 Customization and Use Version 1 Release 2*, SC24-6040.

OpenSSH for z/OS

OpenSSH for z/OS, Program Number 5655-M23, is an official version of a prior porting of OpenSSH to OS/390 UNIX. OpenSSH is a suite of popular UNIX connectivity tools providing secure remote login, remote shell, and ftp between SSH client and server. OpenSSH has been released in the z/OS 1.6 time frame, but actually requires z/OS 1.4 with PTFs to run. OpenSSH is described in Chapter 10, “z/OS OpenSSH” on page 235.

The OpenSSH reference book is *z/OS IBM Ported Tools for z/OS User’s Guide*, SA22-7985.

2

RACF Security Server enhancements

This chapter explains the new features of z/OS RACF from z/OS 1.3 through z/OS 1.6. We discuss the new features of the UNIX System Services with RACF, PADS enhancements, the new dynamic updates to the Class Descriptor Table and RACF Templates, and we provide an explanation of SAF Trace. Other RACF enhancements such as MLS and Password Enveloping are discussed in Chapter 3, "Multilevel Security and RACF" on page 61; and Chapter 6, "RACF Password Enveloping and z/OS LDAP Change Log" on page 149.

2.1 z/OS UNIX System Services Security and RACF

In this section we introduce the capability of complementing the POSIX permission bits protecting a z/OS UNIX resource with an access control list (ACL) for more granular access control. The decisions made by RACF when inspecting the File Security Packet are also explained.

Note that, in this chapter, unless explicitly specified not to apply to zFS, all references to HFS apply to zFS as well.

2.2 HFS ACLs

HFS files and directories are currently protected by POSIX permission bits, contained within the File Security Packet (FSP), also located in the HFS data set. That is to say that the permission bits are not kept in RACF profiles, and, incidentally, are transportable with the file or directory. The simple table below outlines the permissions categories for the file owner, for the group owning the file, and for everyone else.

Table 2-1 Permissions categories

Owner	Group	Other
read write execute	read write execute	read write execute

With the permission bits we cannot permit or restrict access to specific users or groups, which is felt to be a constraint when attempting to establish fine-grained access control. The optional Access Control Lists (ACLs), as introduced in z/OS 1.3, are based on the traditional approach of associating to a resource a list of users and groups who can access the resource, in the mode (read, write, execute, or none) specified. As for the permission bits, they are contained within the file system, as opposed to being stored in RACF profiles. The ACL, if present, is provided to RACF, along with the permission bits, and are taken into consideration if the permission bits do not grant access. With ACLs, the access control algorithm behaves very much like the one RACF uses with the profiles access lists.

ACLs are managed using the specific UNIX commands setfacl and getfacl, but in order to use those one must either have UID(0), or be the file owner or have at least read access to the SUPERUSER.FILESYS.CHANGEPERMS profile in the UNIXPRIV class. These are actually the same requirements as for changing the permission bits.

ACLs can contain a maximum of 1024 entries, each composed of:

- A type: User or group
- An identifier: UID or GID
- The permissions to be given: Read, write or execute

ACL operational support

For ACL operational support:

- The ACLs support inheritance: We can establish a default or model ACL on a directory, which is automatically applied to new files and directories located in this directory. We can also have separate sets of defaults, one used for files and another used for directories.
- The ACL is automatically deleted if the file is removed.
- Several UNIX commands and ISHELL are updated to support ACLs.

2.2.1 Managing ACLs

The reference book to use for the complete syntax of the commands shown in this section is *z/OS UNIX System Services Command Reference*, SA22-7802.

The getfacl command

The **`getfacl`** command displays the file name, the file owner and owning group identities, the base POSIX permissions in *ACL format*, and the ACL entries, if they exist, as shown below:

- #file: MyFile
- #owner: JON
- #group: RACFACL
- user: rwx
- group: r
- other: r
- user: PEGGY: rwx
- group: RACFACL: r-x

The setfacl command

The **`setfacl`** command allows us to perform the operations described below. **`setfacl`** can also be used to change base permissions by omitting the user or group name qualifier, and therefore allows to have one single command for permission granting.

- Modify an ACL, or create it if it does not already exist:

  ```
  setfacl -m user:peggy:rwx,group:racfacl:r-x MyFile
  ```

- Delete an ACL entry:

  ```
  setfacl -x user:peggy:rwx MyFile
  ```

- Delete an entire ACL:

  ```
  setfacl -D MyFile
  ```

- Replace an ACL or create it if it does not exist:

  ```
  setfacl -s u:pekka:r-x,user::rw, group::r-x,other::--- MyFile
  ```

- Set an ACL from the contents of a file:

  ```
  setfacl -S /u/jon/friends MyFile
  ```

  This allows use of *named ACLs*:

  ```
  $ls /u/jon/acls
  writers
  project_managers
  editors
  ```

  In this scenario, the named ACLs will be files containing ACL entries as you would type them into the **`setfacl`** command, one per line; for example, user:camillia:r-x. The file can be primed with the contents of an existing ACL:

  ```
  getfacl  /usr/lpp/java > /u/jon/JavaAcl
  ```

- An ACL can be set from stdin and thus piped in from a **`getfacl`** command:

  ```
  getfacl YourFile | setfacl -S - MyFile
  ```

Other UNIX commands that support ACL

There are various other UNIX commands that support ACLs:

- The **ls** command indicates the existence of ACLs, with the plus sign (+) at the end of the permission bits:

  ```
  ls -l MyFile
  -rw-rwxr--+ 1 BPXROOT SYS1
  ```

- Find is updated to find all files with an ACL of a given type or types:

  ```
  find / -acl a (access ACL only)
  find / -acl a -o -acl d -o -acl f (any ACL type)
  ```

- Find files with ACL entries for a specific user/group:

  ```
  find / -acl_user peggy (finds entry in an ACL type)
  find / -acl_group racfacl (finds entry in an ACL type)
  find / -acl_entry d:u:pekka (look only for directory default entries)
  ```

- Find files with a certain number of ACL entries:

  ```
  find / -acl_count +50
  ```

- **cp** (copy) and **mv** (move/rename) can preserve the file's ACLs across the copy/move operation.
- **pax** and **tar** will support archiving ACLs along with files.
- The **df** command (display file system information) indicates whether the security product in use of the file system supports ACLs.
- **getconf** (get configuration values) reports on whether the security product supports ACLs and the maximum number of ACLs.
- **sh** and **tsch** provide options to test for ACLs on files.

Application Program Interfaces

Various Application Program Interfaces provide support for ACLs, Language Environment® (LE), and REXX provide C functions to manipulate ACLs. The Low level Logical File System (LFS) interface is also available.

RACF also provides callable services such as ck_access (RRSKA00, Check Access) and r_setfacl (IRRSCL00, Unix Access Control List) to handle the ACLs.

2.2.2 ACLs and RACF

RACF Access Checking with ACLs takes into account the base POSIX permission bit and the existing ACL. The full algorithm is rather complex and is shown in "Examples of ACLs use" on page 12, where it is used to illustrate examples. In any case the book to refer to for additional information is *z/OS Security Server RACF Security Administrator's Guide*, SA22-7683, and especially the Appendix F, which provides an extensive check list for identifying access problems.

The ACLs are used for access decisions only if the RACF FSSEC class is active, though **setfacl** can be used to create ACLs at any time, even when FSSEC is inactive.

Similar to RACF profile checking, user entry permissions take precedence over group entry permission, and if more than one group entry applies, the highest access is taken.

The basic file checking algorithm is used so each item below is checked in the specified order and checking stops at the first relevant item:

- Check *owner* bits.

- Check user ACL entries.
- Check union of group bits and group ACL entries.
- All entries are checked until a *single* entry grants the requested access.
- Check *other* bits.

Note: There is a graphical representation of the RACF decisions tree in Figure 2-1 on page 13. We describe below profiles that are important contributors to the decisions made by RACF.

SUPERUSER.FILESYS

SUPERUSER.FILESYS is a profile in the UNIXPRIV class to allow non UID(0) users to get extra privileges regarding handling of files and directories belonging to other users. Depending on the access a user is granted to this profile:

- A user with READ access to the profile can read any local file, and read or search any local directory.
- A user with UPDATE access can write to any local file, and includes privileges of READ access.
- A user with CONTROL access can write to any local directory, and includes privileges of UPDATE access.

RESTRICTED.FILESYS.ACCESS

If the UNIXPRIV profile RESTRICTED.FILESYS.ACCESS if defined, then “other” bits are treated in the same way that UACC, GAT, and ID(*) are treated for users with the RESTRICTED attribute in their USER profile. That is, users with the RESTRICTED attribute cannot be granted file access via the “other” bits, whether or not an ACL exists.

If an exception to this rule has to be set for a specific RESTRICTED user, give a READ permission to RESTRICTED.FILESYS.ACCESS to this user or one of the groups it is connected to. Note that in this case the privilege, if any, provided by SUPERUSER.FILESYS in the UNIXPRIV class, applies to this RESTRICTED user.

Refer to Figure 2-1 on page 13 for a graphical representation of the RACF decisions tree.

SUPERUSER.FILESYS.ACLOVERRIDE

If the UNIXPRIV profile SUPERUSER.FILESYS.ACLOVERRIDE is defined, then SUPERUSER.FILESYS authority is treated like the OPERATIONS attribute during RACF data set profile checking; that is, it prevents users from using their SUPERUSER.FILESYS authority to access file system resources if the users are specifically unauthorized to access through the ACL.

For exception cases, the user or group has to be permitted to SUPERUSER.FILESYS.ACLOVERRIDE with whatever access level is also required for SUPERUSER.FILESYS.

SUPERUSER.FILESYS.ACLOVERRIDE is checked only if an ACL entry match was found for the user or one of the groups the user is connected to, and no ACL entry granted the requested access. If SUPERUSER.FILESYS.ACLOVERRIDE does not exist, or there was no matching entry, SUPERUSER.FILESYS is checked as it is today.

Note: The SUPERUSER.FILESYS.ACLOVERRIDE profile introduces to the HFS files and directories access control an OPERATIONS-like optional behavior, that is reproducing RACF behavior when dealing with a requestor with the OPERATIONS attribute. As such It only makes sense to check this profile if a “denying” ACL entry is encountered during ACL processing. If you simply define this profile with UACC(NONE), then you have created a system switch to enforce the OPERATIONS-like behavior. This can be overridden for individual users/groups by granting the access that would have been required for SUPERUSER.FILESYS.

If you do not wish to implement this behavior, then you have no need to define the SUPERUSER.FILESYS.OVERRIDE profile (and you will save some path length by not doing so).

2.2.3 Auditing and reporting

We can set LOGOPTIONS (ALWAYS) on the FSSEC class to audit on all ACL modifications. There are now two new SMF Type 80 event codes for setfacl (75) and delfacl (76). For setfacl, one audit record is created for each added, modified, and deleted ACL entry. For delfacl only one record is created.

The ICH408I message and file access SMF record are updated, to indicate if an ACL entry was used to determine permissions, failure due to inaccessibility of ACL, or because a user is RESTRICTED.

2.2.4 Migration considerations

The FSSEC class is used for auditing changes to the File Security Packet. It does not need to be active and, prior to z/OS 1.3, its activation had no effect on RACF behavior. Beginning with z/OS 1.3, activating the class also activates the use of ACLs in UNIX file authority checks. Be aware that this class could have been inadvertently activated in the past and you did not notice. Do not activate this class on z/OS 1.3, or later, until you are ready to use ACLs.

In a sysplex, ACLs can be defined within a file system from an uplevel; that is, at least a z/OS 1.3 sysplex node. If the file system is accessed from a downlevel node, however, the ACLs will be received but ignored. If the shared file system is mounted on a downlevel node, the node will not be able to pass the ACLs to the requestor. We recommend migrating all sysplex nodes to z/OS V1R3 before using ACLs, or at least to apply the compatibility APARs (OW50655 for SAF and OW49334 for RACF to downlevel nodes with FSSEC active. The compatibility APARs will force access failures for files with ACLs, except for the file owner or a superuser, when the FSSEC class is active.

2.2.5 Examples of ACLs use

The purpose of these examples is to take you through the RACF access decisions tree shown in Figure 2-1 on page 13, focusing on peculiar decisions that would not be that obvious to first time users of the HFS ACLs.

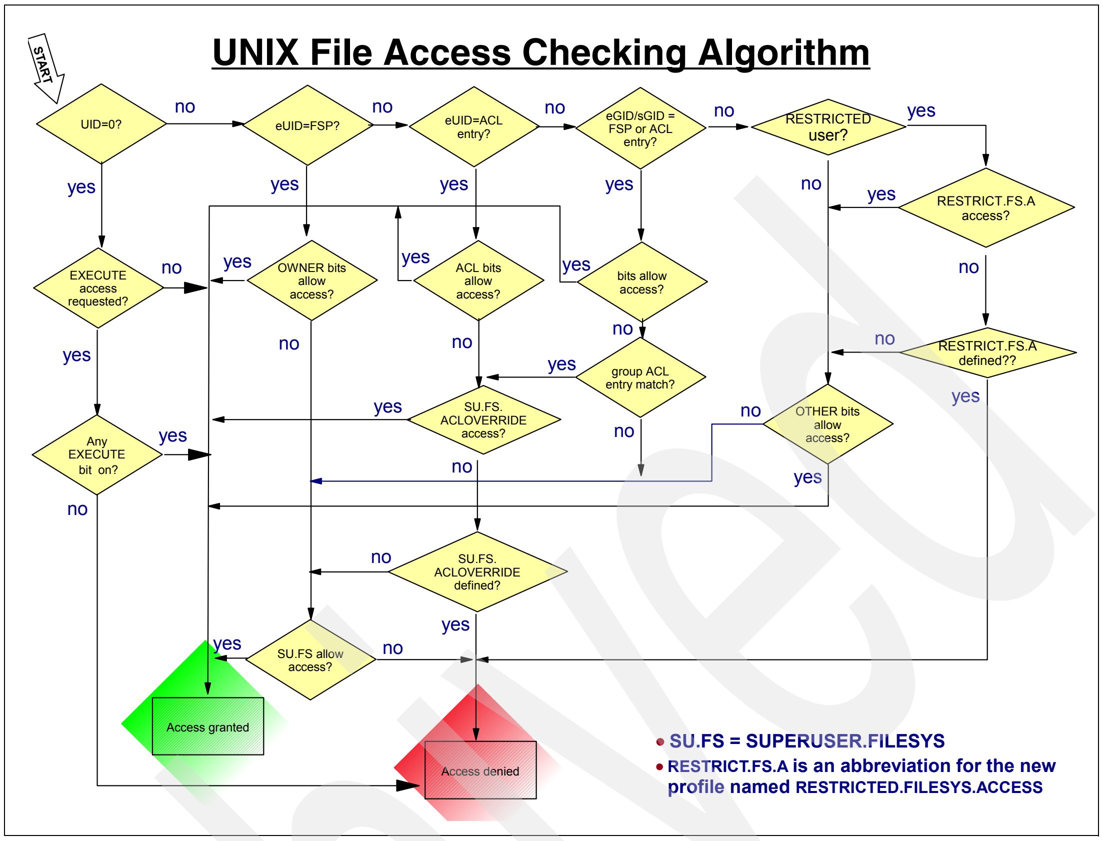


Figure 2-1 HFS ACLs decisions tree

Attention: We are not showing in Figure 2-1 other conditions influencing access to HFS or zFS files, independently from the use of ACLs. Namely:

- A trusted or privileged task has superuser privileges, even if it is not running with a UID>0.
- A superuser with UID(0) does not get execute access to a file if there is not at least one execute permission in the complete path to the file.
- Beginning with z/OS 1.5, security labels can also control access to the file.

Example 1

Remember that in order to take advantage of ACLs you must activate the class FSSEC with the SETROPTS command:

```
SETROPTS CLASSACT(FSSEC)
```

In this example we show how to set up a simple ACL on a directory. First of all we defined a group called ADMIN, then we logged on to UNIX as a superuser, and use the **`setfacl`** command to give ADMIN group read and write access to the /etc directory:

```
setfacl -m group:admin:rw- /etc
```

Now when we use the **`ls -Fla`** command on the /etc directory we get the + character displayed, indicating that the directory does have an ACL:

```
drwxr-xr-x+ 15 OEKERN    OMVSGRP     8192 Aug 18 07:11 /SYSTEM/etc
```

We can also use the **getfacl** command to show us the FSP attributes, as shown in Figure 2-2.

```
(MV08)-/SYSTEM/etc-(OEKERN)-(134):getfacl /SYSTEM/etc
#file:  /SYSTEM/etc/
#owner: OEKERN
#group: OMVSGRP
user::rwx
group::r-x
other::r-x
group:ADMIN:rw-
```

Figure 2-2 getfacl output

To complete this example we try to create a file within the /etc directory, but we access is denied. So now we connect ourselves to the group ADMIN and try again, and this time we can create a file, as we are members of the group ADMIN.

Example 2

In this example we look at how to use the new UNIXPRIV class profile SUPERUSER.FILESYS.ACLOVERRIDE.

As it is shown in the decisions tree above and in the example below, the use of this profile has to be very well understood. As already mentioned, it can be used to have the HFS ACLs checking behavior very much like RACF data sets access control with users with the OPERATIONS attribute.

We created the file test.override in the /etc/ directory. We then changed all the permission bits so no one could get access via the permission bits. We then used the **setfacl** command to have an ACL on the file, and granted the users PUBLIC and CHRIS execute access.

The group ADMIN is given read and write access via the ACL. The **setfacl** commands were:

```
setfacl -m group:admin:rw- /etc/test.override
setfacl -m user:chris:--x /etc/test.override
```

The owner of this file is JON and the file owning group is OMVSGRP.

The **ls -Fla /etc/test.override** command displays the following permissions:

```
----------+  1 JON      OMVSGRP       22 Sep  1 10:50 /etc/test.override
```

The ACL entry displayed using the **getfacl /etc/test.override** command is shown in Figure 2-3.

```
#file:  /etc/test.override
#owner: JON
#group: OMVSGRP
user::---
group::---
other::---
user:CHRIS:--x
group:ADMIN:rw-
```

Figure 2-3 ACL entry display from getfacl command

PEGGY and JON are connected to the group ADMIN and therefore have read and write access to test.override via the ACL.

We define the SUPERUSER.FILESYS profile in UNIXPRIV, with a UACC none and permit CHRIS with CONTROL to this profile, meaning that CHRIS can read or write any local file or directory:

```
RDEF UNIXPRIV SUPERUSER.FILESYS UACC(NONE)
PE SUPERUSER.FILESYS CL(UNIXPRIV) ID(CHRIS) AC(CONTROL)
```

To summarize the file accesses we specified for /etc/test.override:

- JON is the owner of the file. The ACL specifies a RW access for him via his membership to the group ADMIN.
- PEGGY has RW access via her membership to the group ADMIN.
- CHRIS has all accesses via his CONTROL permission to the SUPERUSER.FILESYS UNIXPRIV profile.
- We will also be using the user PEKKA in the example. PEKKA is not a specific user to the file and does not have access either via the permission bits or the ACL.

All users have a UID>0.

Without SUPERUSER.FILESYS.ACLOVERRIDE being defined

We log on as PEGGY and indeed PEGGY can edit the file as she is a member of the group ADMIN.

We then log on as JON but cannot get access to the file, although JON is a member of the ADMIN group just like PEGGY. Actually the reason, as shown in the decisions tree in Figure 2-1 on page 13, is that JON is the owner of the file and the permission bits do not give access to the owner, and therefore ACL checking is not performed.

We log on as CHRIS and, as expected, CHRIS has read/write access to the file.

With SUPERUSER.FILESYS.ACLOVERRIDE defined

Now we define the UNIXPRIV profile SUPERUSER.FILESYS.ACLOVERRIDE:

```
RDEFINE UNIXPRIV SUPERUSER.FILESYS.ACLOVERRIDE UACC(NONE)
```

We try again to edit the file as CHRIS and we are denied access. Following the RACF decision tree in Figure 2-1 on page 13, we can see that:

1. CHRIS is not granted read/write access to the file either via the permission bits or the specified ACL.
2. This leads to the test of the SUPERUSER.FILESYS.ACOVERRIDE profile. The profile has been defined with a UACC(NONE), and CHRIS is not in the access list of the profile.
3. The test for permission to the SUPERUSER.FILESYS UNIXPRIV profile is then bypassed and, despite his CONTROL permission to this profile, CHRIS is barred access to the file because of SUPERUSER.FILESYS.ACLOVERRIDE being defined.

If we want CHRIS to again have the equivalent privilege to a CONTROL access to SUPERUSER.FILESYS, we have to give him CONTROL access to SUPERUSER.FILESYS.ACLOVERRIDE:

```
PE SUPERUSER.FILESYS.ACLOVERRIDE CLASS(UNIXPRIV) ID(CHRIS) AC(C)
```

We can also grant PEKKA and JON CONTROL access to the SUPERUSER.FILESYS.ACLOVERRIDE profile:

```
PE SUPERUSER.FILESYS.ACLOVERRIDE CLASS(UNIXPRIV) ID(PEKKA JON) AC(C)
SETR RACLIST(UNIXPRIV) REFRESH
```

We now log on as PEKKA, but as it was previously the case for JON, PEKKA does not get access, even though he is in the SUPERUSER.FILESYS.ACLOVERRIDE access list with CONTROL. The reason for PEKKA not gaining access is that he does not have any permission bit matching, he is not in the ACL list either as a user or in the group ADMIN, and he is not permitted to SUPERUSER.FILESYS, so the ACL checking throws him out.

JON is still failing to get access as per the decisions tree not considering permission to SUPERUSER.FILESYS.ACLOVERRIDE in his case (that is, being the owner of the file without having any access and still no permission to SUPERUSER.FILESYS).

Recommendation

SUPERUSER.FILESYS.ACLOVERRIDE operates in a way similar to SUPERUSER.FILESYS. Its intended purpose is, as the name implies, to override ACLs that are not granting access, for users permitted to the profile. The whole point though is to not permit any users, or at least exceptionally few, to the profile.

The real planning consideration is that if you do implement SUPERUSER.FILESYS.ACLOVERRIDE, and you do grant an exception by permitting a user to the profile, then you need to keep this user's access level in synchronization with its permission to SUPERUSER.FILESYS, because you do not know whether a given file will have an access list, or whether the user will be on it (if not, SUPERUSER.FILESYS is checked; if so, SUPERUSER.FILESYS.ACLOVERRIDE is checked).

2.3 UNIX identity management

POSIX standard does not require UIDs and GIDs to be unique, and neither does RACF; however, we do recommend that you keep them unique to avoid access ambiguities. With z/OS V1R4, RACF can help allocate a unique UID when creating new UNIX users. These functions have been rolled back to OS/390 2.10 with APAR OW52135.

Note: The functions we describe in this section require the RACF database to be at Application Identity Mapping stage 2 or 3. The AIM stage conversion is explained in *z/OS Security Server RACF System Programmer's Guide*, SA22-7681.

SHARED.IDS

The first facility is a system-wide switch that prevents assigning an already used UID tor GID to a new UNIX user or new UNIX group. This is the SHARED.IDS profile in the UNIXPRIV class. This profile, once defined, stops users other than users with SPECIAL to create shared UID/GIDs:

```
RDEFINE UNIXPRIV SHARED.IDS UACC(NONE)
SETROPTS RACLIST(UNIXPRIV) REFRESH

ADDUSER NEWUSER OMVS(UID(111))
IRR52174I Incorrect UID 111.  This value is already in use by PEKKA.
```

A user without SPECIAL can create a shared UID if the creating user is permitted to the SHARED.IDS profile and uses the SHARED keyword in the OMVS segment:

```
PE SHARED.IDS CL(UNIXPRIV) ID(CREATOR) AC(R)
SETROPTS RACLIST(UNIXPRIV) REFRESH

AU NEWUSER OMVS(UID(111) SHARED)
```

A user with SPECIAL does not have to be permitted to SHARED.IDS; however, she still needs to specify the SHARED keyword to allow sharing of UIDs.

AUTOUID/AUTOGID

RACF can be asked to allocate a new UID or GID, which is not used yet, by using the keyword AUTOUID or AUTOGID when defining a user OMVS segment:

```
AU NEWUSER OMVS(AUTOUID)
IRR52177I User NEWUSER was assigned an OMVS UID value of 4.
```

We can use the APPLDATA of the new BPX.NEXT.USER profile in the FACILITY class to derive candidate UID or GID values. We specify in the APPLDATA a range of numeric numbers, and RACF automatically assigns to the user a unique UID within this range, or a unique GID for a group. For instance, we can fix the ranges 30000–50000 for UIDs, and the range 700-800 for GIDs:

```
RDEFINE FACILITY BPX.NEXT.USER APPLDATA('30000-50000'/700-800)
SETR RACLIST(FACILITY) REFRESH

AU NEWUSER OMVS(AUTOUID)
IRR52177I User NEWUSER was assigned an OMVS UID value of 30000.
AU NEWUSER OMVS(AUTOUID)
IRR52177I User NEWUSER2 was assigned an OMVS UID value of 30001.

AU NEWGRP OMVS(AUTOGID)
IRR52177I Group NEWGRP was assigned an OMVS GID value of 700.
```

We can also NOAUTO in the APPLDATA of BPX.NEXT.USER, which stops the automatic UID or GID assignment.

Specific considerations for RRSF

In an RRSF environment where user updates are kept synchronized across the network, you want to avoid UID/GID collisions when AUTOUID/AUTOGID is used on multiple nodes, for instance, AUTOUID in one node propagating a value by chance already taken on another node. This is accomplished by specifying unique ranges of values in the BPX.NEXT.USER APPLDATA for each node, thus making sure the various nodes do not derive the same value as another node.

If you are using RRSF automatic command direction for the FACILITY class, you must use the ONLYAT keyword, even when changing the profile on the system on which you are logged on. ONLYAT tells RACF not to propagate the command outbound so that profiles on other nodes are not wiped out. Note that the AT keyword is not sufficient, since it will still be subject to propagation. Figure 2-4 shows how ONLYAT is used.

```
RALT FACILITY BPX.NEXT.USER UACC(NONE) APPLDATA('100-5000/NOAUTO')
ONLYAT(NODEA)
RALT FACILITY BPX.NEXT.USER UACC(NONE) APPLDATA('5001-15000/50-500')
ONLYAT(NODEB)
```

Figure 2-4 Sample RDEFINE command for BPX.NEXT.USER

RACF SEARCH enhancement

We can use a new subparameter at the end of the USER or GROUP SEARCH command stating the searched UID or GID. This is also available in the RACF ISPF panels, when using option S or 9 in the User or Group Profile Services panels. The response returned by SEARCH is shown in Figure 2-5 on page 18.

```
SEARCH CLASS(USER) UID(0)
JON
YURI
OMVSID
SEARCH CLASS(USER) UID(30000)
NEWUSER
SEARCH CLASS(GROUP) GID(50)
OMVSGRP
```

Figure 2-5 SEARCH command to find a specific UID or GID

Re-allocating already existing shared UIDs or GIDs

As mentioned at the beginning of this chapter, the RACF database is required to be at AIM stage 2 or 3 for the functions we explained to work.

If you have just gone through the AIM migration or intend to do so in the near future, you may want to detect the duplicate UID and GIDs already existing in your RACF database. Figure 2-6, Figure 2-7 on page 19,and Figure 2-8 on page 19 show the steps required to unload the RACF database and how to use IRRICE to check for duplicate UIDs and GIDs.

```
//JONAA    JOB   (REDBOOK),'JONATHAN BRIGGS',TIME=1440,MSGLEVEL=(1,1),
//    NOTIFY=&SYSUID,MSGCLASS=H
//IRRDBU00 EXEC  PGM=IRRDBU00,PARM='NOLOCK'
//SYSPRINT  DD   SYSOUT=*
//INDD1     DD   DSN=SYSM.RACF.PRIMARY,DISP=SHR
//OUTDD     DD   DSN=JON.RACF.IRRDBU00,DISP=(,CATLG),
//             VOL=SER=08W002,SPACE=(CYL,(25,5)),
//             DCB=(LRECL=4096,RECFM=VB),UNIT=3390
```

Figure 2-6 Sample IRRUDB00 job

The above JCL will unload the RACF database in to a dataset, allowing us to use that as input for the IRRICE report tool and check for duplication. Below is a sample IRRICE job that is extracting record type 0270 (UIDs) and outputting to a dataset for us to view and to sort to check on duplicate UIDs.

```
//JONAA    JOB   (REDBOOK),'JONATHAN BRIGGS',TIME=1440,MSGLEVEL=(1,1),
//    NOTIFY=&SYSUID,MSGCLASS=H
//SEPARATE  EXEC PGM=SORT
//SORTIN    DD   DISP=SHR,DSN=JON.RACF.IRRDBU00
//SORTWK01  DD   SPACE=(CYL,(20,5,0)),UNIT=SYSALLDA
//DS$1      DD   DISP=(NEW,PASS),DSN=&TEMP1,UNIT=SYSALLDA,
//          SPACE=(CYL,(1,1,0))
//DS$2      DD   DISP=(NEW,PASS),DSN=&TEMP2,UNIT=SYSALLDA,
//          SPACE=(CYL,(1,1,0))
//DS$3      DD   DISP=(NEW,PASS),DSN=&TEMP3,UNIT=SYSALLDA,
//          SPACE=(CYL,(1,1,0))
//DS$4      DD   DISP=(NEW,PASS),DSN=&TEMP4,UNIT=SYSALLDA,
//          SPACE=(CYL,(1,1,0))
//DS$5      DD   DISP=(NEW,PASS),DSN=&TEMP5,UNIT=SYSALLDA,
//          SPACE=(CYL,(1,1,0))
//DS$6      DD   DISP=(NEW,PASS),DSN=&TEMP6,UNIT=SYSALLDA,
//          SPACE=(CYL,(1,1,0))
//DS$7      DD   DISP=(NEW,PASS),DSN=&TEMP7,UNIT=SYSALLDA,
//          SPACE=(CYL,(1,1,0))
//DS$8      DD   DISP=(NEW,PASS),DSN=&TEMP8,UNIT=SYSALLDA,
//          SPACE=(CYL,(1,1,0))
//SYSOUT    DD   SYSOUT=*
//SORTOUT   DD   DSN=JON.RACF.IRRICE.UIDS,DISP=(,CATLG),
//          SPACE=(TRK,(20,5,0)),UNIT=3390
//SYSIN     DD *
 SORT    FIELDS=(10,8,CH,A)
 INCLUDE COND=(5,4,CH,EQ,C'0270')
 OPTION  VLSHRT
//
```

Figure 2-7 Sample IRRICE job

```
SORT    FIELDS=(19,10,CH,A)
INCLUDE COND=(5,4,CH,EQ,C'0120')
OPTION  VLSHRT
```

Figure 2-8 Sample SYSIN for GIDs

Wa can then can re-allocate UID and GIDs before we start to use automatic UID/GID assignment.

2.3.1 Enhancements to the OMVS RACF segment

As applications become 64-bit enabled, more legacy code will need to be able to run in AMODE 64. Unix System Services need to manage a 64-bit storage and its limits.

In z/OS V1R6, the RACF OMVS segment will support 64-bit storage management, with a specification of non-shared memory maximum and shared memory maximum. ADDUSER and ALTUSER have been amended to include new keywords of the OMVS segment.

- MEMLIMIT - Maximum bytes allowed for non-shared memory; the maximum value is 15T.
- SHMEMMAX - Maximum bytes allowed for shared memory; the maximum value is 16383P.

The format of data is a numeric value between 1 and 16777215 followed by M, G, T, or P, with:

- M = Megabyte
- G = Gigabyte
- T = Terabyte
- P = Petabyte

2.4 RACF enhancements

In this section we discuss RACF enhancements.

2.4.1 PADS enhancement

Program control, control of program access to data sets (PADS), and protection of programs with sensitive data and algorithm inside (EXECUTE control) are not new to users of RACF. However, experience of RACF clients, and requirements to extend program-based controls to new types of resources and environments made it obvious that the implementation of program-related controls would have to be enhanced. The overhaul of the existing program controls led to the introduction of ENHANCED program security mode in RACF at z/OS 1.4 level, while the traditional, pre-z/OS 1.4, implementation of program controls has been reintroduced as BASIC program security mode. Additionally, some of pre-z/OS 1.4 restrictions have been relaxed in BASIC mode. At lastly, to make transition from BASIC to ENHANCED mode smooth, RACF offers ENHANCED WARNING mode.

The mode of program control is managed through a new FACILITY class profile, IRR.PGMSECURITY, using the APPLDATA field to store the selected mode. The initial definition of the profile can be without an APPLDATA specification:

```
RDEF FACILITY IRR.PGMSECURITY
```

In z/OS 1.4 and later, if the profile is not defined at all, or defined as shown above, the mode is set to BASIC. With the defined IRR.PGMSECURITY profile, switching between the three program control modes is simple—just use the RALTER command with the desired APPLDATA specification and then refresh the PROGRAM class:

```
RALT FACILITY IRR.PGMSECURITY APPLDATA('BASIC'/'ENHANCED'/'ENHWARN')
SETR WHEN(PROGRAM) REFRESH
```

The distinctions between all modes are provided in the output of the SETROPTS LIST command (pre-z/OS 1.4 below is included for a better visibility).

Table 2-2 Distinctions between modes

Program control mode	First line of the SETRLIST LIST output
Pre-z/OS 1.4	ATTRIBUTES = INITSTATS WHEN(PROGRAM)
BASIC	ATTRIBUTES = INITSTATS WHEN(PROGRAM -- BASIC)
ENHANCED	ATTRIBUTES = INITSTATS WHEN(PROGRAM -- ENHANCED)
ENHANCED WARNING	ATTRIBUTES = INITSTATS WHEN(PROGRAM -- ENHANCED WARNING)

This part of the book discusses the BASIC and ENHANCED modes with regards to PADS, as well as ENHANCED WARNING mode for the migration purpose; examples will be provided. All necessary details can be found in *z/OS Security Server RACF Security Administrator's Guide*, SA22-7683, in the chapter "Protecting Programs."

PADS

We focus on these sections on the new program security modes in the context of PADS. The same requirements are applicable to program access to SERVAUTH resources (exploited by TCP/IP) and EXECUTE-controlled programs. We leave the discussion of these two to the readers, given that the topic is very well documented in the reference book we used *z/OS Security Server RACF Security Administrator's Guide*, SA22-7683, in the chapter "Protecting Programs."

So, what is program access to data sets (PADS) about? It is about a particular user being able to open a protected data set only if running a particular program. PADS is about the ability of a task running with a specific user ID to open an otherwise forbidden data set in a specially designated protected program, loaded into a controlled (clean) environment.

Thus, all PADS requires in BASIC mode is the following two ingredients: A clean environment, and a program eligible to open an otherwise forbidden protected data set.

A clean environment, the first part in the PADS equation, consists of controlled programs. A program is said to be controlled if the following two conditions are met:

- A profile, covering this program, was defined in the PROGRAM class.
- PROGRAM control was activated via the SETROPTS WHEN(PROGRAM) command in RACF.

In order to get access to a controlled program, an eligible user ID or group name is to be in the access list of the corresponding PROGRAM class profile.

The system checks on the controlled status of programs via RACF when loading them into storage via LINK, LOAD, XCTL, or ATTACH. Programs loaded via other means will not be checked for PROGRAM Control.

Note: Programs that reside in the UNIX file system are excluded from this discussion. Execution of programs in the UNIX file system is controlled using UNIX security controls (as opposed to RACF PROGRAM profiles), and programs resident in the UNIX file system cannot be used for PADS or program access to SERVAUTH resources.

This restriction also applies to programs written in the TSO/E CLIST language, PERL, Java™, or other interpreted languages that RACF and z/OS cannot protect.

RACF and z/OS can protect programs written in REXX only if they are compiled and link-edited as load modules or program objects.

The second part in the PADS equation is a protected program that physically opens a protected data set. Before z/OS 1.4, a name of the program had to be explicitly specified in the conditional access list for the defined DATASET profile via the WHEN(PROGRAM) specification. In z/OS 1.4 and later you either do the same, or specify the program that invokes (LINK, ATTACH) the OPENing program in the WHEN(PROGRAM) conditional access list. There is no need to put the name of the actually OPEN'ing program into the conditional access list.

An example of program security in basic mode

We developed a simple example of the sample code in "PADS enhancement sample code" on page 290, which performs as follows:

- A program application consists of two programs, ATTAMAIN and ATTATASK; the program ATTAMAIN calls (via the LINK macro) program ATTATASK, which issues the OPEN macro to start processing of the YURY.TEST.PADS data set. Both programs reside in the load library YURY.TEST.LOAD. The JCL to submit the application is shown in Figure 2-9.

```
//YURY1M   JOB (ACCT),'KRITCHEVER',NOTIFY=&SYSUID,MSGCLASS=H,
//      TIME=1440,MSGLEVEL=(1,1),REGION=1M
//ST1  EXEC PGM=ATTAMAIN
//STEPLIB  DD DSN=YURY.TEST.LOAD,DISP=SHR
//INPUTDS  DD DISP=SHR,DSN=YURY.TEST.PADS
//
```

Figure 2-9   Sample job, used in further discussion of PADS-related enhancements

- User YURY1 is not intended to have access to the contents of the data set YURY.TEST.PADS, protected by the corresponding DATASET profile.
- An installation needs to process the YURY.TEST.PADS data set under user YURY1, using the outlined program application, but the data set remains forbidden to user YURY1 at all other times.

The preparatory steps are listed in Table 2-3.

Table 2-3   Preparatory steps

Steps	**RACF commands**
1. Define DATASET profile for load library YURY.TEST.LOAD.	AD ‘YURY.TEST.LOAD’ UACC(NONE)
2. Permit user YURY1 access to the load library.	PE ‘YURY.TEST.LOAD’ ID(YURY) ACC(READ)
3. Define DATASET profile for YURY.TEST PADS.	AD ‘YURY.TEST.PADS’ UACC(NONE)
4. Define PROGRAM class profile for all programs in load library YURY.TEST.LOAD with names, starting from ATTA.	RDEF PROGRAM ATTA* UACC(NONE) ADDMEM('YURY.TEST.LOAD'//NOPADCHK)
5. Define a separate PROGRAM class profile for program ATTATASK because this program OPENs the protected data set. See the note below.	RDEF PROGRAM ATTATASK UACC(NONE) ADDMEM('YURY.TEST.LOAD'//PADCHK)
6. Permit user YURY1 access to the both programs.	PE ATTA* CL(PROGRAM) ID(YURY1) ACC(READ) PE ATTATASK CL(PROGRAM) ID(YURY1) ACC(READ)
7. Refresh the PROGRAM class.	SETR WHEN(PROGRAM) REFRESH

Note: PADS requires that the PROGRAM profile protecting the program accessing the dataset have the exact name of the program; and asterisk (*) is not permitted.

Notice that program ATTAMAIN is controlled, owing to step 4. Since ATTAMAIN is used as a job step program in our JCL, that is the first one to run, and the environment is considered to be clean.

Nonetheless, if the program application is submitted immediately after the above setup, it imminently fails with the messages shown in Figure 2-10 on page 23.

```
ICH408I USER(YURY1   ) GROUP(OMVSGRP ) NAME(###################)
  YURY.TEST.PADS CL(DATASET ) VOL(O8W001)
  INSUFFICIENT ACCESS AUTHORITY
  ACCESS INTENT(READ   )  ACCESS ALLOWED(NONE   )

IEC150I 913-38,IFG0194E,YURYA,ST1,INPUTDS,503E,O8W001,YURY.TEST.PADS
```

Figure 2-10   Data set violation message

To fix this, one needs to specify user YURY1 in the conditional access list for the DATASET profile for YURY.TEST.PADS; the following PERMIT command does just that:

```
PE 'YURY.TEST.PADS' ID(YURY1) ACC(READ) WHEN(PROGRAM(ATTAMAIN))
```

Now, if the exact same job is submitted again, it runs to the successful completion.

```
ICH70001I YURY1    LAST ACCESS AT 10:01:56
$HASP373 YURY1M    STARTED - INIT 1    - CLA
+YURY -- WE ARE IN ATTAMAIN
+YURY -- WE ARE IN SUBTASK
+YURY -- IN LOADED MODULE
+YURY -- RECNUM = 01358
-                                         -
-JOBNAME  STEPNAME PROCSTEP    RC   EXCP
-YURY1M            ST1         00     18
-YURY1M   ENDED.  NAME-KRITCHEVER
$HASP395 YURY1M    ENDED
```

Figure 2-11   Output, produced by the sample job, running to its normal completion

Note that any ATTA* program loaded from the load library YURY.TEST.LOAD may call ATTATASK to open the protected data set YURY.TEST.PADS, bringing some "anonymity" to which program established the controlled environment. The need arises now for a more strict control of which programs are establishing the clean environment on which PADS to be used, hence the implementation of an enhanced Program Security mode.

Program security enhanced mode

Welcome the ENHANCED mode. No longer does an anonymously controlled program establish a clean environment for PADS—it must be established by a specifically designated program. This program must be covered by a PROGRAM class profile, defined with the APPLDATA specification. Two possible values that may be specified in the APPLDATA field are MAIN and BASIC. If APPLDATA is not coded, a clean environment is not sufficiently "clean" for PADS in ENHANCED mode.

So, to satisfy PADS requirements, the following simple rules are to be followed when choosing between MAIN and BASIC:

- Programs that the user executes in batch directly using JCL, and programs that the user executes through the TSO/E TSOEXEC command or IKJEFTSR callable service in a TSO/TMP address space (batch or foreground) can be defined as "MAIN" in the PROGRAM profile APPLDATA field. APF authorized programs that are loaded from APF authorized load libraries and added to the corresponding AUTHCMD or AUTHPGM list in IKJTSOxx can be defined MAIN as well. Meaning that these are programs you will use for PADS and that you are trusting to maintain the environment correctly.

You should not specify MAIN for programs invoked in other ways because RACF does not honour the MAIN attribute in other cases.

- Programs, establishing the trusted environment for PADS, that the user executes under TSO, but not through TSOEXEC or IKJEFTSR, can be defined as BASIC.

Note: You must not define the TSO/E terminal monitor program (TMP) or any part of it (such as IKJEFT01, IKJEFT1A, IKJEFT1B, IKJEFT02), or ISPF or any part of it (such as ISPF, ISPSTART, ISPMAIN) as MAIN or BASIC. They provide too much of a generalized environment controlled by a user and are not pointing at a well-defined set of operations.

Let us see what happens if the mode is changed from BASIC (or pre-z/OS 1.4) to ENHANCED and the PROGRAM class setup remains the same as above.

```
- Switch mode to ENHANCED

   RALT FACILITY IRR.PGMSECURITY APPLDATA('ENHANCED')
   SETR WHEN(PROGRAM) REFRESH
- Run the same job as before; the setup remains the same:

   ICH70001I YURY1    LAST ACCESS AT 12:20:48
   $HASP373 YURY1M    STARTED - INIT 1    -
   +YURY -- WE ARE IN ATTAMAIN
   ICH426I NON-MAIN PROGRAM IS IN CONTROL. CONDITIONAL ACCESS LIST
         BYPASSED FOR DATA SET YURY.TEST.PADS
   ICH408I USER(YURY1   ) GROUP(OMVSGRP ) NAME(####################)
     YURY.TEST.PADS CL(DATASET ) VOL(O8W001)
     INSUFFICIENT ACCESS AUTHORITY
     ACCESS INTENT(READ   )  ACCESS ALLOWED(NONE   )
   ICH428I PROGRAM ATTAMAIN FROM YURY.TEST.LOAD ESTABLISHED THE CURRENT
           EXECUTION ENVIRONMENT
   IEC150I 913-38,IFG0194E,YURY1M,ST1,INPUTDS,503E,O8W001,YURY.TEST.PADS
   IEA848I NO DUMP WAS PRODUCED FOR THIS ABEND, DUE TO SYSTEM OR
           INSTALLATION REQUEST
   IEF450I YURY1M ST1 - ABEND=S913 U0000 REASON=00000038
```

Figure 2-12 Example of PADS-related violations in ENHANCED mode

Message ICH426I explains why user YURY1 may no longer access data set YURY.TEST.PADS, and message ICH428I helps to identify the causer of the problem by pointing to program ATTAMAIN.

What should have been done here is:

- Assign a PROGRAM profile to ATTAMAIN, with the exact name of the program.
- Since the ATTAMAIN program executes in a batch job and no TSO/E TSOEXEC command or IKJEFTSR callable service is involved, we need to define the program as MAIN in the new PROGRAM class profile.

```
- RDEF PROGRAM ATTAMAIN ADDMEM('YURY.TEST.LOAD'//NOPADCHK) APPLDATA('MAIN’)
- PE ATTAMAIN CL(PROGRAM) ID(YURY1) ACC(READ)
- SETR WHEN(PROGRAM) REFRESH
```

Figure 2-13 Commands to define a job step program as MAIN in ENHANCED mode

Now the same job executes successfully. This job successfully executes, even if the ATTAMAIN program is defined as BASIC; however, we strongly recommend that you follow the above rules of choosing between MAIN and BASIC, since BASIC is lessening the controls done by RACF.

What we have shown so far is the direct migration from BASIC to ENHANCED program security mode. If an installation had only a couple of programs that should have been redefined, then it would not be too difficult to migrate to ENHANCED mode. Regrettably, in real life the number of programs that are to be redefined may be quite large; so, some other identification methods, rather than executing many jobs and analyzing the results, would have to be exploited.

The section “Migrating from BASIC to ENHANCED program security mode” in the chapter “Protecting Program” of *RACF Security Administrator’s Guide*, SA22-7683, provides the comprehensive discussion of how the transition is to be accomplished. The initial step, recommended by the book, is to download the RACF database, using the IRRDBU00 utility, and then examine the type 0402 (Data Set Conditional Access) records. Figure 2-14 on page 26 is an example of such a procedure.

```
- Run the IRRDBU00 job

   //YURYU   JOB (ACCT),'YURY K.',NOTIFY=&SYSUID,MSGCLASS=H,
   //   TIME=1440,MSGLEVEL=(1,1),REGION=2M
   //UNLOAD   EXEC  PGM=IRRDBU00,PARM=NOLOCKINPUT
   //SYSPRINT DD    SYSOUT=*
   //INDD1    DD    DISP=SHR,DSN=SYSM.RACF.PRIMARY
   //OUTDD    DD    DSN=YURY.RACFDB.UNLOAD,
   //           DISP=(,CATLG),UNIT=SYSALLDA,
   //           SPACE=(8196,(10000,2000),RLSE),
   //           DCB=(RECFM=VB,LRECL=4096,BLKSIZE=8196)
   //

- Process the output using the following sample job

   //YURYS   JOB (ACCT),'YURY K.',NOTIFY=&SYSUID,MSGCLASS=H,
   //   TIME=1440,MSGLEVEL=(1,1),REGION=2M
   //SEPARATE  EXEC PGM=SORT
   //SORTIN    DD   DISP=SHR,DSN=YURY.RACFDB.UNLOAD
   //SORTWK01  DD   SPACE=(CYL,(5,5,0)),UNIT=SYSALLDA
   //SYSOUT    DD   SYSOUT=*
   //SORTOUT   DD   DSN=YURY.TEST.SORTOUT,DISP=(NEW,CATLG,DELETE),
   //          SPACE=(TRK,(15,5)),UNIT=SYSALLDA
   //SYSIN     DD *
    SORT    FIELDS=(10,44,CH,A,55,6,CH,A)
    INCLUDE COND=(5,4,CH,EQ,C'0402',AND,
                  62,7,CH,EQ,C'PROGRAM')
    OPTION  VLSHRT
//

- Examine the output data set using installation-dependent tools; here is the example of
a type 0402 record in the produced output data set

----+----1----+----2-....-5----+----6----+----7----+----8----+----9----+----0
0402 YURY.TEST.PADS  ....  08W001 PROGRAM  ATTATASK YURY1    READ     00000

(see RACF Macros and Interfaces (SA22-7682-03), Chapter 10 “RACF Database Unload Utility
(IRRDBU00) Records”, for the format of the DSCACC record format)
```

Figure 2-14 Analysis of conditional access lists in RACF database for migration purpose

For each program found, the following needs to be determined:

- Who executes the program—users or some other programs
- Intended method of execution (batch, TSO)
- For TSO only—whether the user can use the TSO/E TSOEXEC command or IKJEFTSR callable service

This process produces an initial list of MAIN and BASIC candidates. The list helps to RDEFINE new specific PROGRAM profiles or use RALTER to add APPLDATA('MAIN') or APPLDATA('BASIC') as required. With the new or updated PROGRAM class profiles in place it is the right time to switch BASIC mode to ENHANCED WARNING mode, which we mentioned in the very beginning of this section. While there is no immediate need to re-IPL the system after the switch and do some testing, we recommend to do so as soon as it is possible, for there are always active address spaces that might have already passed PADS-related checks in BASIC (or pre-z/OS 1.4) mode.

Figure 2-15 shows what happens when the job in ENHANCED WARNING mode with no PROGRAM class profiles is defined with the APPLDATA specification.

```
- Switch mode to ENHANCED WARNING

   RALT FACILITY IRR.PGMSECURITY APPLDATA('ENHWARN')
   SETR WHEN(PROGRAM) REFRESH

- Submit the job; only the ATTA* profile in the PROGRAM class exists. It is the joblog
of the completed job:

   ICH70001I YURY1    LAST ACCESS AT 14:43:51
   $HASP373 YURY1M    STARTED - INIT 1    - CLASS A - SYS MV08
   +YURY -- WE ARE IN ATTAMAIN
   ICH427I NON-MAIN PROGRAM IS IN CONTROL. TEMPORARY USE OF CONDITIONAL
          ACCESS LIST ALLOWED FOR DATA SET YURY.TEST.PADS

   ICH428I PROGRAM ATTAMAIN FROM YURY.TEST.LOAD ESTABLISHED THE CURRENT
           EXECUTION ENVIRONMENT
   +YURY -- WE ARE IN SUBTASK
   +YURY -- IN LOADED MODULE
   +YURY -- RECNUM = 01358
   -                                         --TIMINGS (MINS.)--            ----PAG
   -JOBNAME  STEPNAME PROCSTEP    RC   EXCP    CPU    SRB  CLOCK   SERV  PG   PAGE
   -YURY1M            ST1         00     18    .00    .00     .0     92   0      0
```

Figure 2-15 Switching to ENHANCED WARNING mode for gathering data for further migration

RACF immediately points to the violation that would normally lead to ABEN913 if the mode was ENHANCED; however, in ENHANCED WARNING mode, the job runs to its successful completion and messages ICH427I and ICH428I help to identify causers of possible problems. We recommend that you collect and analyze the messages; they will allow you to fine-tune the PADS setup in ENHANCED mode.

The final step is the actual migration to ENHANCED mode. During installation you will have to watch for possible problems, and then fix them by adjusting PROGRAM profiles or reversing to ENHANCED WARNING mode via the IRR.PGMSECURITY profile.

Dynamic templates

The RACF database template describes the format of data stored in the RACF profiles.

In z/OS releases prior to z/OS 1.5, a copy of the templates is shipped in source form in SYS1.MODGEN(IRRTEMP1), exists on the RACF database, and is built in system memory at IPL time.

The IRRMIN00 utility, which creates and updates RACF databases, writes the templates on the database. After a change was made to the templates on the RACF database, the system had to be re-IPLed in order to update the in-storage copy of the template.

z/OS V1R5 introduces the dynamic templates support, which:

- Automatically builds the in-storage templates from the latest level of templates at IPL time. At z/OS 1.5 the templates are shipped as a module in compiled format as IRRTEMP2. It is linked into two load modules, the RACF initialization load module and the IRRMIN00 utility load module, so that they both can determine the latest level of the templates.

- Allows an administrator to bring the in-storage profiles up to the latest level, without a re-IPL, when service is applied that has templates changes but does not otherwise require an IPL. This function is performed by IRRMIN00 PARM=ACTIVATE.
- Prevents IRRMIN00 PARM=UPDATE from downleveling the templates on the database.
- Prevents IRRMIN00 PARM=NEW from wiping out the active RACF database.

The IRRMIN00 utility is documented in *z/OS Security Server RACF System Programmers Guide*, SA22-7681.

IPLing the uplevel system when the templates have changed

Normally, the RACF database should be updated to new release or service level templates by using IRRMIN00 PARM=UPDATE before IPLing the system. If the templates have not been updated, RACF will automatically build the correct level of the in-storage templates at IPL time and issue an ICH579E message, shown in Figure 2-16.

.

```
ICH579E RACF TEMPLATES ON DATABASE ARE DOWNLEVEL:
        OA03853 00000010.00000010; USING TEMPLATES AT LEVEL
        HRF7709 00000023.00000010 FROM IRRTEMP2.
        RUN IRRMIN00 PARM=UPDATE.
```

Figure 2-16 ICH579E message

The message contains:

- The level of the templates from the database
- The level of the in-storage templates that were built from IRRTEMP2

In Figure 2-16, the database had some service applied to it (OA03853). It is at release HRF7708 (00000010) and service level (00000010). The release level is updated for each new release. The service level is updated each time a fix is made to the templates in the RACF database. The latest level of the templates, HRF7709 00000023, 00000010 was the level built in storage.

Note: While the system will continue to authenticate users and perform access checks with the correct in-storage level of the templates, the templates on the RACF database are still downlevel. Any utilities, such as IRRUDB00, that use the templates on the database instead of the in-storage templates will produce inconsistent results until IRRMIN00 PARM=UPDATE is run.

You can find out the release/service level of the in-storage copy of the templates by issuing the SET LIST command, as shown in Figure 2-17 on page 29.

```
RACF SET LIST
IRRH005I ((RACF)) RACF SUBSYSTEM INFORMATION:
   TRACE OPTIONS                  - NOIMAGE
                                  - NOAPPC
                                  - NOSYSTEMSSL
                                  - NORACROUTE
                                  - NOCALLABLE
                                  - NOPDCALLABLE
                                  - NODATABASE
                                  - NOASID
                                  - NOJOBNAME
   SUBSYSTEM USERID               - SUPUSER
   JESNODE (FOR TRANSMITS)        - MOPVS08
   AUTOMATIC COMMAND DIRECTION IS *NOT* ALLOWED
   AUTOMATIC PASSWORD DIRECTION IS *NOT* ALLOWED
   PASSWORD SYNCHRONIZATION IS *NOT* ALLOWED
   AUTOMATIC DIRECTION OF APPLICATION UPDATES IS *NOT* ALLOWED
   RACF STATUS INFORMATION:
         TEMPLATE VERSION         - HRF7709 00000023.00000010
         DYNAMIC PARSE VERSION    - HRF7709
```

Figure 2-17 Dynamic templates support - Output of the SET LIST command

Note: The output from the SET LIST command displays the in-storage templates version. It does not show what the actual database templates level is.

As shown in Figure 2-17, the in-storage copy of the templates is at the new release (HRF7709), the release level has been incremented (00000023), and the service level is (00000010). So you can now run the IRRMIN00 utility to dynamically update the templates in the database. This allows us to use ALL RACF utilities, as well as carry on users' authentications and performing access checks.

Figure 2-18 shows the JCL used to run the IRRMIN00 program.

```
//JONAA    JOB   (REDBOOK),'JONATHAN BRIGGS',TIME=1440,MSGLEVEL=(1,1),
//    NOTIFY=&SYSUID,MSGCLASS=H
//STEP1    EXEC PGM=IRRMIN00,PARM=UPDATE
//SYSPRINT DD SYSOUT=*
//SYSRACF  DD DISP=SHR,DSN=SYSM.RACF.PRIMARY
```

Figure 2-18 IRRMIN00 JCL

The IRRMIN00 job generates a listing of the templates that are applied, as shown in Figure 2-19 on page 30.

```
IRR8005 Beginning RACF data base initialization.
$/VERSION HRF7709 00000023.00000010
$*
$/TEMPLATE 001 GROUP VERSION 1
$/SEGMENT  001 BASE
GROUP    001 00 00 00000000 00 TEMPLATE FOR A GROUP PROFILE
ENTYPE   002 00 00 00000001 01 ENTRY TYPE
VERSION  003 00 00 00000001 01 TEMPLATE VERSION NUMBER
SUPGROUP 004 00 80 00000008 FF SUPERIOR GROUP
AUTHDATE 005 00 20 00000003 FF GROUP CREATION DATE
AUTHOR   006 00 80 00000008 FF OWNER OF GROUP
INITCNT  007 00 00 00000002 FF RESERVED FOR FUTURE USE
 -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  817 Line(s) not Displayed
CDTCASE  019 00 00 00000001 00 CDT - Case of profile names
$*                                   X'00' UPPER case
$*                                   X'80' ASIS - preserve case
$/END
IRR8012 RACF data base updates complete.
```

Figure 2-19   IRRMIN00 job output (abbreviated)

Changing templates via PTF

In the past when we have applied PTFs that have amended or updated the templates, we have been required to IPL to bring the new templates into storage from the updated templates in the RACF database. Now it is sufficient to just run the IRRMIN00 utility with PARM=ACTIVATE.

Running IRRMIN00 PARM=ACTIVATE against the active RACF database

The scenario here is that we have received an APAR/PTF with some new field or segment to be added to RACF, and we therefore need to apply these to our system. We have said that we can run IRRMIN00 with PARM=ACTIVATE to dynamically update the in-storage templates. Assuming that we forgot to run IRRMIN00 PARM=UPDATE to update the database templates with the new templates shipped in IRRTEMP2, we get the message shown in Figure 2-20.

```
IRR8032 PARM=ACTIVATE specified, but the level of the database templates: HRF7709
00000023.00000010,
is not higher than the level of templates on the system: HRF7709 00000023.00000010.
```

Figure 2-20   Output from IRRMIN00 - Back level database templates

Once that we have applied the APAR/PTF, and ran IRRMIN00 with PARM=UPDATE, we can now run IRRMIN00 with PARM=ACTIVATE. This time the IRRMIN00 job works and we receive the message shown in Figure 2-21.

```
IRR8026 PARM=ACTIVATE specified; IRRMIN00 is preparing to activate the templates
HRF7709 00000023.00000011.
IRR8027 IRRMIN00 has finished activating the templates.
```

Figure 2-21   Output from IRRMIN00, successful ACTIVATE

Running IRRMIN00 PARM=NEW against the active database

With the dynamic templates support, it is also not possible to wipe out an active RACF database by running IRRMIN00 PARM=NEW against it (PARM=NEW formats a non-VSAM

DASD data set as a RACF database). Attempting to launch IRRMIN00 with PARM=NEW against the activated database yields the messages shown in Figure 2-22.

```
IRR8005 Beginning RACF data base initialization.
IRR8024 PARM=NEW specified for active RACF database. Processing stopped.
IRR8004 RACF data base initialization terminated in error.
```

Figure 2-22 Output from IRRMIN00 PARM=NEW against the active RACF database

2.4.2 Dynamic CDT

It is worth beginning with a reminder about the Class Descriptor Table (CDT). The CDT is a table in RACF that defines general resource class names and attributes. IBM supplies a CDT in the module ICHRRCDX. The classes entries here are not to be modified by clients. There is another module called ICHRRCDE, also called the *installation-defined CDT*, which can be customized by clients who wish to create and maintain their own classes' descriptions. You can add classes when, for instance, adding new CICS® regions. Some vendor products require you to add classes to the CDT to define the specific resources their products access.

Prior to z/OS 1.6, a change in the installation-defined CDT requires an IPL to complete the update. z/OS 1.6 introduces the Dynamic CDT support that makes this IPL not required anymore.

Important: Note that a change to the IBM-provided CDT still requires an IPL to be taken into account.

The Dynamic CDT support involves using a new IBM general resources class named *CDT* to create a class definition, with a CDTINFO segment to define class attributes. The Dynamic CDT, shown in Figure 2-23 on page 32, is built in its own data space with the SETROPTS RACLIST(CDT) command. The other parts of the CDT still exist in common storage, copied from modules ICHRRCDX and optional ICHRRCDE (these parts are labeled on the chart as *static CDT*). The Dynamic CDT is a logical extension of the static CDT.

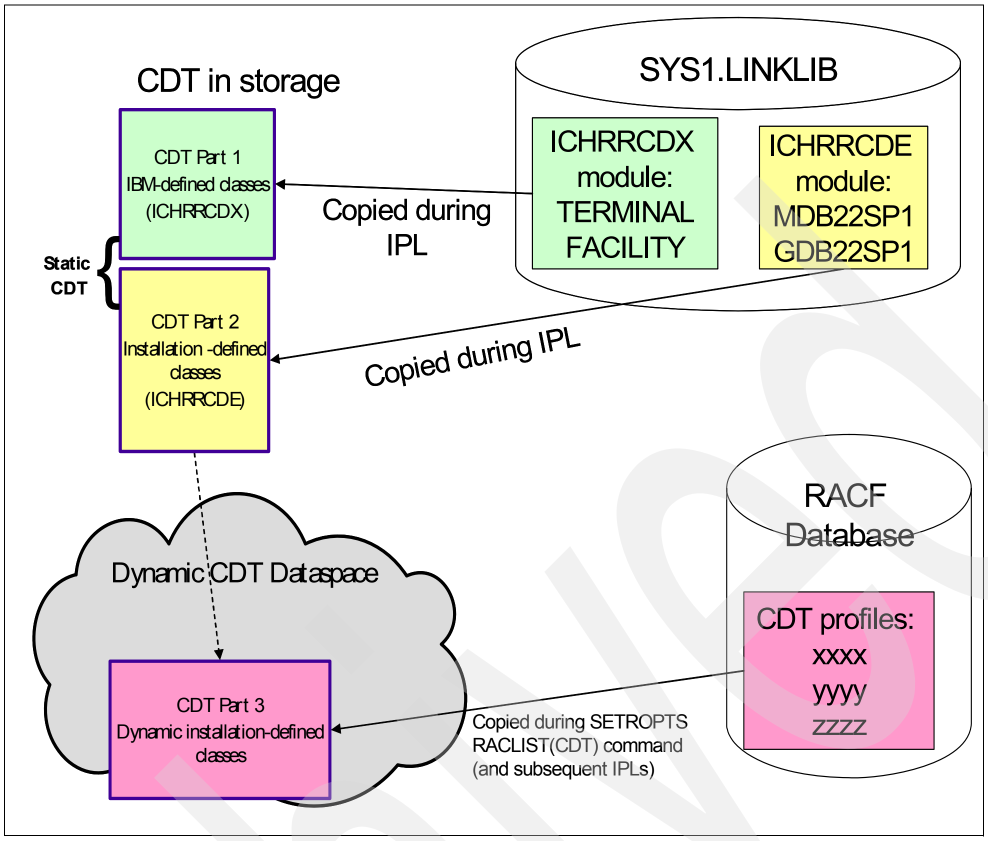

CDT in storage
SYS1.LINKLIB
CDT Part 1
IBM-defined classes
(ICHRRCDX)
ICHRRCDX module: TERMINAL FACILITY
ICHRRCDE module: MDB22SP1 GDB22SP1
Copied during IPL
Static CDT
CDT Part 2
Installation -defined classes
(ICHRRCDE)
Copied during IPL
RACF Database
Dynamic CDT Dataspace
CDT profiles: xxxx yyyy zzzz
CDT Part 3
Dynamic installation-defined classes
Copied during SETROPTS RACLIST(CDT) command (and subsequent IPLs)

Figure 2-23 Dynamic CDT implementation

The static CDT can still be updated as before by updating assembler module ICHRRCDE. The contents, however, only become effective after a system IPL. The Dynamic CDT can be updated by using RACF commands to create profiles in the CDT class and do not require an IPL.

Example of a dynamic class definition

To define a new class to the installation CDT we use profiles in the new CDT class for defining the class and the segment CDTINFO to define the class attributes. The naming conventions for a RACF class have been relaxed. Class names can now be 1–8 characters, and they should include at least one character of the following:

- 0–9
- # (X'7B')
- @ (X'7C')
- $ (X'5B')

In our example we define the new class @JONBOY:

```
RDEFINE CDT @JONBOY UACC(NONE) CDTINFO(DEFAULTUACC(NONE) FIRST(ALPHA) MAXLENGTH(200)
OTHER(ALPHA,NUMERIC) POSIT(301) RACLIST(REQUIRED))
```

The parameters in the CDTINFO segment are shown in Table 2-4 on page 33. These parameters are explained in detail in *z/OS Security Server RACF Command Language Reference*, SA22-7687.

Table 2-4   CDTINFO segments parameters

CDTINFO field	Description
DEFAULTUACC	Specifies the minimum access allowed if the access level is not set when a resource profile is defined in the class. ACEE ALTER CONTROL UPDATE READ NONE
DEFAULTRC	Specifies the return code that RACF will provide from RACROUTE REQUEST=AUTH or RACROUTE=FASTAUTH when RACF and the class are active and, if required, the class has been processed using SETROPTS RACLIST. 0 = The access request was accepted. 4 = No profile exists. 8 = The access request was denied.
CASE	CASE specifies whether mixed case profiles are allowed for the class. UPPER specifies RACF to translate the profile names for the specified class to be in upper case. ASIS specifies that lowercase characters are allowed in any position where alphabetic characters are allowed, based on character restrictions in the FIRST and OTHER keywords.
FIRST	Specifies a character type restriction for the first character of the profile name. One or more of the following may be specified. ALPHA specifies an alphabetic character (A–Z). NUMERIC specifies a digit (0–9). NATIONAL specifies characters # (X’7B’), @ (X’7C’), and $ (X’5B’). SPECIAL specifies any character except the following: ► A blank ► A comma ► A parenthesis ► A semicolon ► Those characters in ALPHA, NUMERIC, or NATIONAL
GENLIST	Specifies whether SETROPTS GENLIST is to be allowed. DISALLOWED is the default. ALLOWED DISALLOWED
CDTINFO	Specifies the name of the class that groups the resources within the class specified by the CLASS operand.
KEYQUALIFIERS	Specifies the number of matching qualifiers RACF uses when loading generic profile names to satisfy an authorization request if a discrete profile does not exist for the resource. The default for KEYQUAL is 0.
MACPROCESSING	Specifies which type of Mandatory Access Control (MAC) processing is required for the class. NORMAL specifies that normal MAC processing is required. If and when a MAC check is performed, the user’s SECLABEL must dominate that of the resource. REVERSE specifies that reverse MAC processing is required. If and when a MAC check is performed, the SECLABEL of the resource must dominate that of the user. EQUAL specifies that equal MAC processing is required. If and when a MAC check is performed, the SECLABEL of the user must be equivalent to that of the resource.

CDTINFO field	Description
MAXLENX	Specifies the maximum length of resource and profile names for this class when a RACROUTE macro is invoked with the ENTITIYX keyword or a profile is added or changed via a RACF command processor.
MAXLENGTH	Specifies the maximum length of a resource and profile names for this class when MAXLENX is not specified.
MEMBER	Specifies the name of the class grouped by the resources within the class specified by the CLASS operand.
OPERATIONS	Specifies whether RACF is to take the OPERATIONS attribute into account when it performs authorization checking. YES is the default. YES NO
OTHER	See description for FIRST.
POSIT	Specifies the POSIT number associated with the class.
PROFILESALLOWED	Specifies whether you want RACF to allow profiles to be defined for this RACF resource class. YES is the default. YES NO
RACLIST	Specifies whether SETROPTS RACLIST is to be allowed for the class. DISALLOWED is the default. ALLOWED DISALLOWED REQUIRED
SECLABELREQUIRED	Specifies whether a SECLABEL is required for the profiles of the specified class when SETROPTS MLACTIVE is on. SECLABELSREQUIRED(NO) means that RACF will not require a SECLABEL for profiles in this class; however, if a SECLABEL exists for this profile and the SECLABEL class is active, RACF will use it during authorization checking. SECLABELSREQUIRED(NO) applies to general resource classes that have no profiles, such as DIRAUTH, or for classes that contain no data, such as OPERCMDS and SECLABEL. SECLABELSREQUIRED(YES) means that RACF will require a SECLABEL for profiles in this class when SETROPTS MLACTIVE is on.
SIGNAL	Specifies whether an ENF signal should be sent to listeners when RACLISTed profiles are created, updated, or deleted for authorization checking.

We now use the SETROPTS RACLIST command to add the Dynamic CDT. In our example the class “JONBOY” will eventually be recognized as a RACF general resource class:

```
SETROPTS CLASSACT(CDT) RACLIST(CDT)
```

A data space is created for the Dynamic CDT, which is a logical extension of the static CDT. If another installation class is created, then the SETROPTS RACLIST REFRESH command needs to be issued to refresh the CDT data space. A regular RLIST CDT JONBOY NORACF CDTINFO command may be used to display the CDTINFO segment. At the next IPL, RACF will remember that the CDT class was RACLISTed and the Dynamic CDT will be built during RACF initialization.

We can use existing RACF commands to administer the new Dynamic CDT resource profiles, and the DSMON utility has been updated to place a D at the side of a class to indicate that this is a dynamic class. The database utility IRRDBU00 has also been updated to include profiles in the dynamic classes.

Note: Applications that scan all classes in the RACF class descriptor table need to be changed in order to access the Dynamic CDT information. Refer to *z/OS Security Server RACROUTE Macro Reference*, S122-7692.

You can migrate your existing installation-defined classes to the Dynamic CDT. A program is available on the RACF Web site to help translate definitions from the ICHRRCDE module into commands to create CDT profiles. This is the CDT2DYN program explained in the next section. During the migration process a duplicate class is allowed to exist both in ICHRRCDE and Dynamic CDT, but the Dynamic CDT takes precedence. As already mentioned, the IBM-supplied CDT cannot be made dynamic, and updates to the module ICHRRCDX are taken into account at the next IPL.

Using the CDT2DYN program

This program is obtained from:

`http://www-1.com/servers/eserver/zseries/zos/racf/goodies.html`

Attention: This program is available on an as-is basis. Please read the disclaimer before starting using it.

The program translates the definitions that z/OS 1.6 RACF loaded into storage from the ICHRRCDE module during IPL, into commands to create the dynamic classes.

The program is a REXX exec. The purpose of this program is to examine the contents of the current classes in the RACF static CDT. For all static installation-defined classes it will create commands necessary to add them to the Dynamic CDT (BUILDCMD parameter). It can then execute the commands that were built (RUNCMD parameter), which will define the dynamic classes.

We downloaded the program into “JON.CDT2DYN.EXEC”, where it can be edited, if needed. We kept the dataset_attributes the same. In Table 2-5 we describe the parameters that can be changed.

Table 2-5 :Parameters that can be changed

Parameters	Description
BUILDCMD	The default is BUILDCMD. if you want to build commands to define dynamic classes from your current installation classes in the CDT.
NOBUILDCMD	If you do not want to build the commands. You can specify this with the RUNCMD parameter if you have already built your commands, optionally edited them, and now you want them to be executed.
RUNCMD	To run the commands built by this utility.
NORUNCMD	The default is NORUNCMD, if you do not want to run the commands built by this utility, as you might want to edit or examine the commands before they are executed.

An output dataset is created and optionally executed. If the dataset already exists, it will be overwritten. Values of RECFM=V or RECFM=F and LRECL=255 must be used for the program to run properly. The output dataset is created with the prefix of the user ID who invokes the program.

We now run the program by issuing the following TSO command:

```
TSO EXEC ‘JON.CDT2DYN.EXEC’
```

The program, while executing, issues the messages shown in Figure 2-24.

```
>>> Starting to build the commands.....
>>> Processing class #YURYCL
>>>
>>> Installation-defined Classes for which commands were built: 1
>>>
>>> Examine commands in JON.DYNCLAS.EXEC for accuracy
>>> and edit them as needed.
>>> Then run the commands to define the Dynamic Classes
>>> by invoking this utility with parameters
>>> NOBUILDCMD and RUNCMD.
>>>
>>> End of processing for CDT2DYN Revision 1.0.00
```

Figure 2-24 Output from the CDT2DYN program

We had created one class in the installation-defined static CDT, called “#YURYCL”, that we processed with CDT2DYN in order to migrate it to a Dynamic CDT. The commands generated by CDT2DYN are stored in the “JON.DYNCLAS.EXEC” dataset as another EXEC. Invoking this JON.DYNCLASS.EXEC to complete the migration to Dynamic CDT yields the messages shown in Figure 2-25.

```
>>>
>>> Issuing command: RDEFINE CDT #YURYCL   CDTINFO( CASE(UPPER) DEFAULTRC( 4 ) D
EFAULTUACC( ACEE ) FIRST( ALPHA NATIONAL NUMERIC ) GENLIST( DISALLOWED ) KEYQUA
LIFIERS( 0 ) MACPROCESSING( NORMAL ) MAXLENX( 39 ) MAXLENGTH( 39 ) OPERATIONS(
NO ) OTHER( ALPHA NATIONAL NUMERIC SPECIAL ) POSIT( 25 ) PROFILESALLOWED( YES )
 RACLIST( DISALLOWED ) SIGNAL( NO ) SECLABELSREQUIRED( NO ) )  <<<
>>> Investigate the following command message. <<<
>>>
>>> Commands run: 1 <<<
>>>
>>> Maximum command return code: 0 <<<
```

Figure 2-25 Output from the DYNCLASS exec

To ensure that the classes have been defined we can use the SEARCH command to confirm they have been defined as profiles in the CDT class:

```
TSO SEARCH CLASS(CDT)
JONBOY
YURYCL
```

2.4.3 RACROUTE traces

A SAF trace has been made available in z/OS 1.2, that can be used to trace:

- RACROUTE invocations
- calls to RACF callable services
- RACF Database Manager requests that go through the RACF routers

The reference book is *z/OS Security Server RACF Diagnosis Guide*, GA22-7689.

We are providing here an example of the SAF trace setup to capture your RACROUTE requests, which also explains how to view the output you will receive.

GTF is used for collecting the information about the RACROUTE calls that the trace captures, as specified in the RACF SET command. Here we use our own GTF procedure.

First of all we needed to create a RACF started task profile for our test GTF trace procedure, with the user ID JON to be the started task user ID:

```
RDEFINE STARTED GTFJON.* STDATA(USER(JON)) UACC(NONE) AUDIT(ALL(READ))
SETROPTS RACLIST(STARTED) REFRESH
```

We then created our own GTF trace procedure, shown in Figure 2-26, and the test job that is performing the RACROUTE requests, as shown in Figure 2-27.

```
//GTFJON  PROC MEMBER=AAA
//IEFPROC EXEC PGM=AHLGTF,PARM='MODE=EXT,DEBUG=NO,TIME=YES',
//  TIME=1440,REGION=2880K
//SYSLIB  DD   DSNAME=SYS1.PARMLIB(&MEMBER),DISP=SHR
//IEFRDER DD   DSNAME=JON.GTF.TRACE,UNIT=SYSALLDA,SPACE=(TRK,20),
//             DISP=(NEW,KEEP)
```

Figure 2-26 Test GTF trace procedure

```
//JONAA    JOB   (REDBOOK),'JONATHAN BRIGGS',TIME=1440,MSGLEVEL=(1,1),
//    NOTIFY=&SYSUID,MSGCLASS=X
//ST1    EXEC  PGM=IEBGENER
//SYSPRINT  DD   SYSOUT=*
//SYSIN     DD   DUMMY
//SYSUT1    DD   DISP=SHR,DSN=SYS1.MACLIB(ENQ)
//SYSUT2    DD   DUMMY
//
```

Figure 2-27 Test job to generate RACROUTE requests

The steps are as follows.

1. We are now starting the GTF trace with the console command S GTFJON.JGTF.
2. We can now enter the RACF SET command to start the SAF trace, to capture information about RACROUTE service type number 1; that is, AUTH (authorization, refer to Table 2-6 below for the different service types):

   ```
   RACF SET TRACE(RACROUTE(TYPE(1)) JOBNAME(JONAA))
   ```

Table 2-6 RACROUTE CALL = service type number

RACROUTE CALL	**Service/type number in hex**	**Service/type number in decimal**
AUTH	1	1
FASTAUTH	2	2
LIST	3	3
DEFINE	4	4
VERIFY	5	5
EXTRACT	6	6
DIRAUTH	7	7
TOKENMAP	8	8

RACROUTE CALL	Service/type number in hex	Service/type number in decimal
VERIFYX	9	9
TOKENXTR	A	10
TOKENBLD	B	11
EXTRACT,BR=YES	C	12
AUDIT	D	13
STAT	E	14
SIGNON	F	15
TOKENMAP,XMEM	10	16
TOKENXTR,XMEM	11	17

> **Note:** We are focusing in this chapter on RACROUTEs tracing. Be aware that tracing is also available for 49 RACF callable services. Additional information about the callable services tracing can be found in the reference book.

3. Once we have entered the above SET command, we need to respond to the WTORs shown in Table 2-28. Our replies are in bold. The reasons why we are getting these WTORs is due to the fact that we coded SYS1.PARMLIB(&MEMBER) in the SYSLIB DD statement of the GTF trace JCL. In a real environment we would have set up an actual member containing:

```
TRACE=USERP
USR=(F44),END
```

```
*12 AHL100A  SPECIFY TRACE OPTIONS
 R 12,TRACE=USRP
 IEE600I REPLY TO 12 IS;TRACE=USRP
 TRACE=USRP
*13 AHL101A  SPECIFY TRACE EVENT KEYWORDS --USR=
 R 13,USR=(F44),END
 IEE600I REPLY TO 13 IS;USR=(F44),END
 USR=(F44),END
 AHL103I  TRACE OPTIONS SELECTED --USR=(F44)
*14 AHL125A  RESPECIFY TRACE OPTIONS OR REPLY U
 R 14,U
 IEE600I REPLY TO 14 IS;U
```

Figure 2-28 WTORs responses

4. We can then start the test job, then stop it, and then also stop the tracing:

```
RACF SET TRACE(NORACROUTE NOJOBNAME)
P JGT
```

> **Note:** It is strongly recommended that you specify NOJOBNAME when shutting down the RACROUTE tracing. Not specifying this parameter may result in extra tracing to be performed the next time the trace is re-started.

5. Now we can examine the output from the GTF trace with the IPCS Dump/trace analysis. From the main panel of the IPCS dialog chose option 0 DEFAULTS, and in the Source field specify the data set that you used to collect the GTF trace. In our case the data set was JON.GTF.TRACE, so this would be coded as follows:

   ```
   Source  ==> DSNAME('JON.GTF.TRACE')
   ```

   Then using option 6 COMMAND we get access to the IPCS subcommand Entry panel, where we type the following command in:

   ```
   GTF USR
   ```

6. With in the IPCS Output Stream panel we can view the GTF trace in great detail, as shown in Figure 2-29 on page 40.

```
Trace Identifier:             00000036
Record Eyecatcher:            RTRACE
Trace Type:                   RACFPRE
Ending Sequence:              ........
Calling address:              00000000  97AD3F4C
Requestor/Subsystem:          DSALSYSO  JES2z105
Primary jobname:              JON
Primary asid:                 0000002A
Primary ACEEP:                00000000  009DE150
Home jobname:                 JON
Home asid:                    0000002A
Home ACEEP:                   00000000  009DE150
Task address:                 00000000  009E6E88
Task ACEEP:                   00000000  009DE150
Time:                         BBAD830B  D54F1349
Error class:                  ........
Service number:               00000001
RACF Return code:             00000000
RACF Reason code:             00000000
Return area address:          00000000  7FFC0B80
Parameter count:              0000000D


Area length:                  00000068

Area value:
00000000  00000000  00680000  00015000  | ..............&. |
17B20406  7FF50604  7F686DBC  00000000  | ...."5.."._..... |
00000000  00000068  00000000  00000000  | ................ |
00400000  00000000  00000000  00000000  | . .............. |
00000000  00000000  00000000  00000000  | ................ |
00000000  00000000  00000000  00000000  | ................ |
00000000  00000000                      | ........         |

Area length:                  0000005C

Area value:
5C000000  08000000  04000000  00000000  | *............... |
00000000  00000000  00000000  00000000  | ................ |
00000000  7F686FBC  00CEB758  00000000  | ....".?......... |
00000000  00000000  00000000  00000000  | ................ |
00000000  00000000  00000000  00000000  | ................ |
7FFC0C3C  17B203F8  00000000            | "......8....     |

Area length:                  00000008

Area value:
D6C6C6E2  C5E30024                      | OFFSET..         |

Area length:                  00000035

Area value:
D4D6D7E5  E2D6F84B  E8E4D9C9  4BE8E4D9  | MOPVSO8.JON.JON |
C9C14BD1  F0F0F0F1  F4F0F94B  C4F0F0F0  | .J0001409.D000 |
F0F1F0F1  4B6F4040  40404040  40404040  | 0101.?           |
40404040  40                            |                  |
Area length:                  00000008
```

Figure 2-29   Sample output from GTF Trace

```
Area value:
D6C6C6E2  C5E30028                     | OFFSET..        |

Area length:               00000009

Area value:
08D1C5E2  E2D7D6D6  D3                 | .JESSPOOL       |

Area length:               00000008

Area value:
D6C6C6E2  C5E30050                     | OFFSET.&        |

Area length:               00000050

Area value:
50018252  55569555  55555555  55555555 | &.b...n........ |
818382B0  B783AD15  BDB18C9C  15151515 | acb..c......... |
818382B0  B783AD15  8381B0B7  928C8215 | acb..c..ca..k.b. |
9C80B68C  918C1515  55555555  55555555 | ....j........... |
BDB18C9C  15151515  8381B0B7  928C8215 | ........ca..k.b. |

Area length:               00000008

Area value:
D6C6C6E2  C5E30054                     | OFFSET..        |

Area length:               0000000E

Area value:
0DE2E8E2  D6E4E340  C3D9C5C1  E3C5     | .SYSOUT CREATE  |

Area length:               000000C0
```

Figure 2-30 Continuation of sample output from GTF Trace

For each traced RACROUTE request, RACF creates two R_TRACE entries: A pre-processing and a post-processing entry, distinguished by the *trace type*. The example of the R_TRACE entry in Figure 2-29 on page 40 is for the pre-processing (type RACFPRE). A post-processing entry would contain trace type RACFPOST; this entry would also contain meaningful RACF return and reason code.

Each trace entry consists of the following four parts.

Header

This provides general information about the request type (service number) and environment at the time of the RACROUTE call. Different addresses in the fields Home ACEEP and Task ACEEP indicate that a task might have created its own security context.

Parameter list

There are always two subentries here. The first one is a SAFP parameter list, and the second subentry represents a parameter list for the particular RACROUTE REQUEST type. In our example it is the ACHKL (RACROUTE REQUEST=AUTH Parameter List).

Parameter specifications

Each parameter specification is represented by two subentries. The first subentry always contains the Area Value of OFFSET; in our example.

D6C6C6E2 C5E30024 means offset +X’24’ into the ACHKL. The second subentry provides the contents of the field, pointed to by the address at the specified OFFSET.

Security context

These are the RACF control blocks (ACEE and ACEE Extension), describing the existing security context.

The RACF Diagnosis Guide publication discusses the RACROUTE Trace option, and provides additional details, which we encourage our readers to look into. Another RACF manual, a RACF Data Areas publication, is to be used as well, for it contains layouts of particular parameter lists.

2.5 RACF digital certificates handling enhancements

In this section we describe the enhancements of the RACDCERT RACF command in terms of new functions and new parameters that have appeared since OS/390 2.8. (The RACDCERT command has already been described as such in the redbook *OS/390 Security Server 1999 Updates Installation and Implementation Guide*, SG24-5629, in the OS/390 2.8 availability time frame.)

Note that the enhancements specifically introduced by z/OS 1.6 are described in a specific section at the end of this chapter.

The functions provided through the RACDCERT command are also tightly related to the internal operations of the z/OS PKI Services. A redbook already addresses the z/OS PKI Services as of z/OS 1.5: *Implementing PKI Services on z/OS*, SG24-6968.

The reference book for the syntax and explanation of the RACDCERT command is *z/OS Security Server RACF Command Language Reference*, SA22-7687. The authorization scheme for using the RACDCERT command did not go under changes, and we refer the reader to the above z/OS publication to find out which conditions and privileges are precisely needed to handle certificates belonging or not to the user.

2.5.1 The PKIX standards

In this section we provide information about the now universally adopted PKIX standards, as these are the standards that RACF certificate management is to abide with, and more generally the standards that all IBM Public Key Infrastructure (PKI) products are following.

Digital certificates have been in use since the late 1980’s; however, the design and implementation of supporting technologies proved to vary from vendor to vendor. One of the major problems encountered when attempting to implement a Public Key Infrastructure was the lack of interoperability between PKI products; hence the lack of potential scalability or capability to aggregate different PKI domains.

This eventually was considered to be a very severe impediment to the development of PKI use in the industry, and to address this issue the Internet Engineering Task Force (IETF) launched the Public-key Infrastructure (PKIX) working group in 1995. Details of the group’s achievements can be found at the IETF Web site:

`http://www.ietf.org`

To make a long story short, here are the standards that a PKIX-compliant PKI must be using today:

- The PKCS#10 format for the certificate request

- The X.509 V3 format for the signed certificate
- The X.509 V2 format for the CRL
- The LDAP protocol to reach the CRL residing in an LDAP directory
- The OCSP (Online Certificate Status Protocol) protocol for a real time access to revocation information

Note: In the context of use of the RACF RACDCERT command, abiding with the PKIX standards means that certificate generation is conforming to the X.509 V3 format, and certificate requests are produced according to the PKCS#10 standard. Likewise, RACF processes received certificate and certificate requests only if they conform to these standards.

The x.509 V3 certificate format

The format of digital certificates evolved from the x.509 V1 format to today's X.509 V3. This evolution was dictated by real-life utilization of the certificates that demonstrated some security exposures and the under-utilization of certain certificate fields, whereas new fields were needed; for instance, fields that could be implemented and used as needed by specific applications. The *extension fields* appeared with x.509 V3, yielding a certificate format as shown in Figure 2-31.

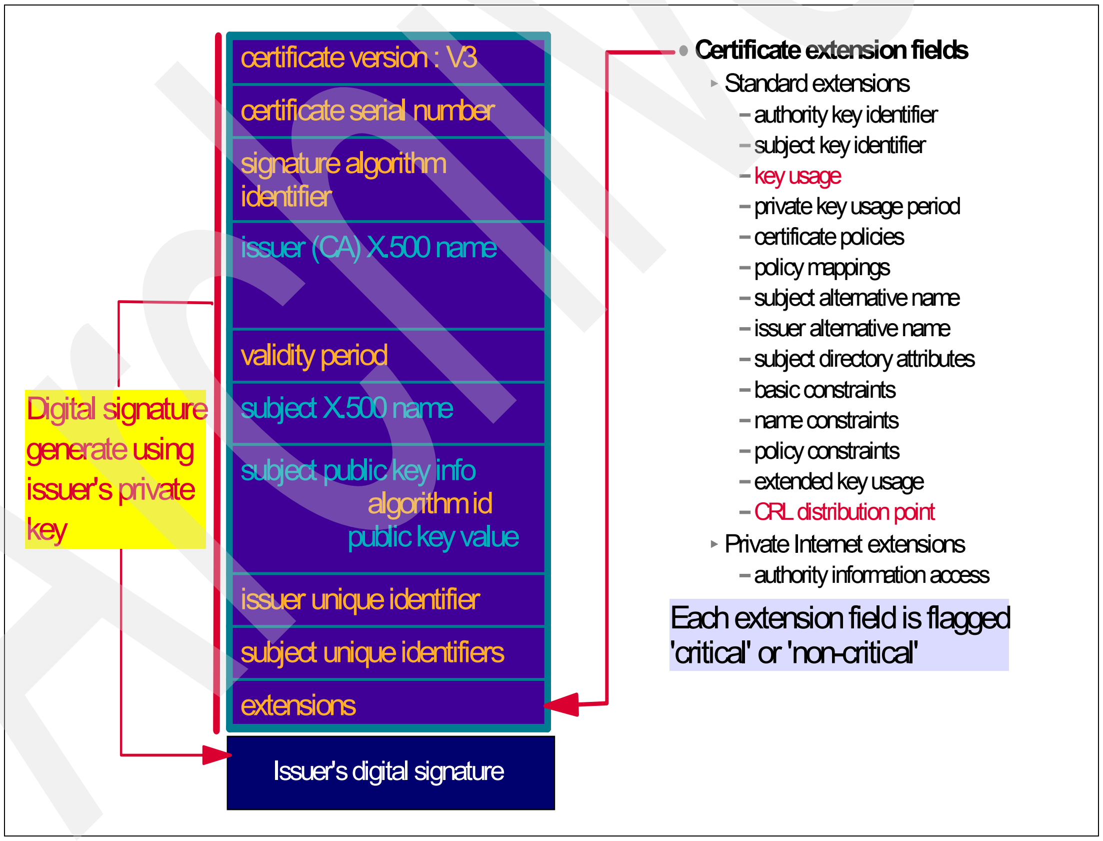


Figure 2-31 X.509 V3 certificate format

The x.509 v3 certificate extension fields

The extension fields are optional fields of which creation, name, and contents are just a matter of convention established with the certificate-exploiting application. An extension is agreed upon in the context of the use of the certificate by a specific application, and it is given a name, a value, and a criticality flag.

The idea of the criticality flag is to state how important it is for the certificate recipient application to recognize the content of the field. If the certificate exploiting application is not able to handle an extension field, then the process is carried on if the extension is marked as non-critical. It is aborted if the extension is marked as critical. Not recognizing an extension marked as critical can potentially lead to severe operating or security exposures.

As a consequence, the structure of a certificate extension consists of the following fields:

Type	Specifies the format of the value field
Criticality	A one-bit flag, used as explained above
Value	The actual extension data

Although the extensions are intended to provide whatever flexibility and meaning an application requires from the contents of the certificate, experience showed that the use of certificate extensions by different real applications were converging in terms of needs and meanings. Therefore, for the sake of interoperability via extension fields standardization, X.503 also specifies the contents of four sets of standard extension fields, which are:

- Extensions with information pertaining to the public key
- Extensions with information pertaining to the use of the certificate
- Extensions specifying attributes for the owner and the certification authority
- Extensions with information about the certificate hierarchy

These fields are described in RFC 3280. Next, we provide a brief overview of their purpose and indicate the recommended settings of the criticality flag.

Extensions with information pertaining to the public key

This section looks at extensions that contain information specifically about the public key.

Authority key identifier (non-critical)

This field indicates the unique identity of the key used by the CA to sign the certificate. This covers the case where the CA has used several signature keys since it was put into operation. The field value is intended to speed up the retrieval process of the correct CA public key by the certificate receiving application.

Subject key identifier (non-critical)

This field indicates the unique identity of the public key enclosed in the certificate. This covers the case where the certificate owner has had, or will have, several keys. It is intended to speed up the retrieval process of the correct private key.

Key usage (critical)

This field specifies the utilization domain of the public key in the certificate. The standardized usages are:

Digital signature	Any signature other than keyCertSign or cRLSign.
Non Repudiation	The subject's public key can be used by a non-repudiation service to verify signature.
keyEncipherment	The subject's public key is used for secure key transport.
dataEncipherment	The subject's public key is used for data confidentiality.
keyAgreement	The subject's public key is used by a key agreement protocol, such as Diffie-Hellman. It must go along with either the encipherOnly or decipherOnly bit.

keyCertSign	The subject's public key is used to verify a signature on a digital certificate.
cRLSign	The subject's public key is used to verify a signature on a certificate revocation list.
encipherOnly	The subject's public key is used by a key agreement protocol, and it can only be used for enciphering data while performing key agreement.
decipherOnly	The subject's public key is used by a key agreement protocol, and it can only be used for deciphering data while performing key agreement.

CRL distribution point (non-critical)

This extension provides the location, usually an LDAP URI, of the Certificate Revocation List to check for the certificate's validity.

Freshest CRL (non-critical)

This is also known as *Delta CRL distribution point*. It identifies, in a syntax similar to the CRL Distribution Point, how delta CRL information is obtained.

Extended key usage (critical or non-critical)

This extension indicates that the subject's public key may be used for purposes other than what is indicated in the key usage extension:

serverAuth	Used for TLS server authentication
clientAuth	Used for TLS client authentication
codeSigning	Used to sign downloadable code
emailProtection	Used to protect e-mail
timeStamping	Used for binding the hash of an object to a time
OCSP signing	Used to sign OCSP responses

Extensions with information pertaining to certificate use

This section looks at extensions that contain information specifically about certificate use.

Certificate policies (critical or non-critical)

This extension contains information about the CA policy under which the certificate has been issued and the purposes for which the certificate may be used. This information is provided as an OID (Object Identifier) number. If the extension is marked critical, then the exploiting application must abide with the designated policy.

Policies Mappings (non-critical)

This extension is used in intermediate CA certificates. It establishes correspondence as designed by the issuing CA. An exploiting application can therefore make a decision about accepting a subject CA policy based on the mapping indicated by the issuing CA.

Extensions with information pertaining to certificate owner and certification authority attributes

This section looks at extensions that contain information specifically about the certificate owner and certification authority attributes.

Subject alternative name (critical if certificate subject field is empty)

This extension enables additional identities to be bound to the subject of the certificate. That is, the subject can be designated by alternate identities such as:

- An Internet e-mail address
- A DNS name
- An IP address
- A uniform resource identifier (URI)

Several alternate identities can be specified in the extension.

Issuer alternative name (critical or non-critical)

This extension enables use of Internet-style identities for the certificate issuer, as with the subject alternative name.

Subject directory attributes (non-critical)

This extension contains attributes and their values related to the identity of the subject. A typical identification attribute in this context is the subject's nationality.

hostIdMapping (critical)

The hostIdMapping extension (OID 1 3 18 0 2 18 1) is an IBM extension, also available for public use. It contains a hostId-userName pair that can be used by an application to directly map the certificate to the specified user ID. The use of hostIdMapping extension by RACF is explained in "The hostIdMapping extension" on page 50.

Extensions with information pertaining to the certificate hierarchy

This section looks at extensions that contain information specifically about the certificate hierarchy.

Basic constraints (critical)

This field indicates whether the subject is an end entity or a CA. If the subject is a CA, it gives the certification path depth (that is, it gives the maximum number of non-self-issued intermediate certificates that may follow this certificate in a valid certification path.

Name constraints (critical)

The name constraints extension is used only in a CA certificate. It indicates a name space within which all subject names in subsequent certificates in a certification path must be located. It is intended to specify acceptable cross-certification domains.

Policy constraints (critical or non-critical)

The policy constraints extension can be used in certificates issued to intermediate CAs. It can be used to prohibit policy mapping or require that each certificate in a path contain an acceptable policy identifier.

InhibitAnyPolicyConstraint (critical)

This extension can be used in certificates issued to intermediate CAs. *Inhibit any policy* indicates that the special anyPolicy identifier is not considered an explicit match for other certificate policies.

2.5.2 Handling of the X.509 V3 extensions

Refer to "The x.509 v3 certificate extension fields" on page 43 for an explanation of the extension fields.

The RACDCERT command for ADD, GENREQ, ALTER, and EXPORT will ignore extensions that are not critical, meaning that these extensions will be copied as is from the original certificate image.

For critical extensions, the only ones that the RACDCERT command allows to be defined in a certificate are in the following list. Attempting to ADD a certificate with critical extension not in this list results in an error message to be generated, and the certificate is not added.

- keyUsage - OID { 2 5 29 15}
- 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 }
- subjectKeyIdentifier - {2 5 29 14}
- authorityKeyIdentifier - {2 5 29 35}

Note that the BasicConstraints extension is examined by RACF when a RACDCERT ADD CERTAUTH function is issued. The certificate is added only if the BasicConstraints field states that this is a CA certificate.

The RACDCERT LIST function displays the KeyUsage and SubjectAltName extensions. These extensions are also copied to the certificate request produced by a RACDCERT GENREQ function.

The hostIdMapping extension is itself ignored when using the RACDCERT command, except for its acceptance in a received certificate as mentioned above. It is actually taken into account only when the certificate is used for authentication.

2.5.3 The RACDCERT GENCERT function

The full syntax of the command is shown in Figure 2-32 on page 48. We focus in this section on the RACDCERT parameters that affects the extension fields in the generated certificate. These parameters are emphasized in bold characters in the figure.

```
RACDCERT [ ID(userid) | MULTIID | SITE | CERTAUTH ]
GENCERT [ (request-data-set-name) ]
   [ SUBJECTSDN(
      [ CN(‘common-name’)] [ T(‘title’)] [ OU(‘organizational-unit-name1’
      [ , ‘organizational-unit-name2’,... ] ) ] [ O (‘organization-name’) ]
      [ L (‘locality’) ] [ SP (‘state-or-province’) ] [ C (‘country’) ] ) ]
   [ SIZE (key-size) ]
   [ NOTBEFORE (
   [ DATE (yyyy-mm-dd) ] [ TIME (hh:mm:ss) ] ) ]
   [ NOTAFTER ( [ DATE(yyyy-mm-dd) ] [ TIME (hh:mm:ss) ] ) ]
   [ WITHLABEL (‘label-name’) ]
   [ SIGNWITH ( [ CERTAUTH„ SITE ]
   LABEL (‘label-name’) ) ]
   [ PCICC | ICSF ]
   [ KEYUSAGE ( [ HANDSHAKE ] [ DATAENCRYPT ] [ DOCSIGN ] [ CERTSIGN ] ) ]
   [ ALTNAME ( IP(numeric-ip-address) DOMAIN(‘internet-domain-name’)
             EMAIL(femail-addressf) URI(funiversal-resource-identifierf) ) ]
```

Figure 2-32 RACDCERT GENCERT syntax

The following RACDCERT GENCERT keywords generate certificates extensions:

- KEYUSAGE

  Depending on the parameter:

  – HANDSHAKE

    This parameter generates the KeyUsage extensions.

    • DigitalSignature
    • keyEnciphermen

  – DATAENCRYPT

    • dataEncipherment

  – DOCSIGN

    • nonRepudiation

  – CERTSIGN

    • keyCertSign
    • cRLSign

- ALTNAME

  This keyword generates the alternative names for a subject or an issuer. The alternate name can be:

  – An IP numeric address in the IPV4 dotted decimal form
  – A fully qualified Internet domain name
  – A fully qualified e-mail address
  – A Universal Resource Identifier (such as http://itso.ibm.com)

- CERTAUTH

  Using the CERTAUTH keyword influences the contents of the following extensions:

  – basicConstraints

    The basicConstraints extension specifies that this is a Certification Authority certificate.

- SIGNWITH

The IssueAltname and the IssuerKeyId extensions are taken from the SubjectAltName and the SubjectKeyId extensions of the signing certificate.

– Note that BasicConstraints, IssuerAltname, and KeyUsage are critical extensions.

Figure 2-33 is an example of RACDCERT GENCERT with generation of certificate extension fields. Figure 2-34 shows the result of performing a RACDCERT LIST against the generated certificate.

```
 ===> racdcert id(patkap) gencert  subjectsdn(cn('pk') ou('pssc') o('ibm') c('fr
'))withlabel('test certificate')signwith(certauth label('racf ca'))
keyusage(handshake dataencrypt) altname(ip(10.11.12.13))
```

Figure 2-33 Generating a certificate with extension fields

```
Label: test certificate
Certificate ID: 2QbXwePSwdejhaKjQIOFmaOJhomDgaOF
Status: TRUST
Start Date: 2005/05/19 00:00:00
End Date:   2006/05/19 23:59:59
Serial Number:
     >OB<
Issuer's Name:
     >CN=racf ca.OU=montpellier.O=ibm.C=fr<
Subject's Name:
     >CN=pk.OU=pssc.O=ibm.C=fr<
Subject's AltNames:
  IP: 10.11.12.13
Key Usage: HANDSHAKE, DATAENCRYPT
Private Key Type: Non-ICSF
Private Key Size: 1024
Ring Associations:
*** No rings associated ***
```

Figure 2-34 Listing a certificate with extension fields

The HIGHTRUST certificate status

THe HIGHTRUST keyword complements the already existing TRUST and NOTRUST keywords used with the RACDCERT ADD and RACDCERT ALTER functions. Remember that the TRUST or NOTRUST status of a certificate is a switch telling RACF whether this certificate, although residing in the RACF database, can be used.

This keyword has been added to discriminate between personal certificates and certification authority certificates. Although CA certificates can still be set with the TRUST status, the user may prefer to give a HIGHTRUST status for those certificates. Note that:

1. Only certificates with a CERTAUTH identify can be given the HIGHTRUST status. A command like the one below applying to a personal (that is, non-CERTAUTH) certificate, is to execute but is putting the target certificate in the TRUST status only:

   ```
   RACDCERT ID(PATKAP) ALTER(LABEL('test certificate')) HIGHTRUST
   ```

2. The HIGHTRUST status of a Certification Authority certificate stored in the RACF database is a requirement when it comes to process a certificate signed by this authority, which contains a hostIdMapping extension, as explained below.

The hostIdMapping extension

The hostIdMapping extension (OID 1 3 18 0 2 18 1) is an IBM extension, also available for public use. RACF automatically maps a valid certificate to the RACF user ID provided in the extension. This mapping is controlled at the server level with the SERVAUTH profile class, and at the RACF policy level with the HIGHTRUST attribute in the CA DIGTCERT profile.

The ASN.1 definition of this extension is shown in Figure 2-35.

```
id-ce-hostIdMapping OBJECT IDENTIFIER ::=  { 1 3 18 0 2 18 1 }

HostIdMapping ::= SEQUENCE {
        hostName            [1] IMPLICIT IA5String,
        subjectId           [2] IMPLICIT IA5String
}
```

Figure 2-35 ASN.1 definition of hostIdMapping

The hostName part is usually a domain name, such as, for example, mvo8.pssc.ibm.com, and the subjectId is the actual RACF user ID to map to.

Note that the choice for a hostName is just a matter of convention—the point being that the same hostName will have to be used in the definition of a RACF profile in the SERVAUTH class, as explained below.

The hostIdMapping Extension is a third method for mapping a certificate to a RACF user ID, the two other methods being:

- A certificate for the user is already installed in the RACF database.
- Or Certificate Name Filtering criteria are in place, which map his certificate to a RACF user ID.

The use of the extension for mapping purposes is tightly controlled:

- The Certification Authority that signed the certificate containing the extension must be installed as CERTAUTH in the RACF database with the HIGHTRUST attribute.
- The application requiring the mapping, via the initACEE callable service (IRRSIA00), must be permitted in READ to a profile in the SERVAUTH class of which name is of the form IRR.HOST.*hostname*.

An example of definition of a SERVAUTH that allows the exploitation of the extensions:

```
RDEF SERVAUTH IRR.HOST.MVO8.PSSC.IBM.COM
```

Permit, for instance, your Web server started task user ID with READ to this profile:

```
PE IRR.HOST.MVO8.PSSC.IBM.COM CL(SERVAUTH) ID(WEBSRV) ACC(R)
```

Now activate and RACLIST the class SERVAUTH, if not already done:

```
SETR CLASSACT(SERVAUTH)
SETR RACLIST(SERVAUTH)
SETR RACLIST(SETRAUTH) REFR
```

Note: The extension can comprise several pairs of hostName-userIds values. RACF is the recipient that scans all the values pairs looking for a host name that matches a permitted SERVAUTH profile name.

Note: The RACDCERT command does not provide the capability of generating a certificate with a hostIdMapping extension; however, this capability is available with the R_PKIServ (IRRSPX00). This callable service is exploited by the z/OS PKI Services.

2.5.4 Hardware assistance to certificate generation

Prior to z/OS 1.4, the RACDCERT GENCERT generated the RSA key pair using software only. At z/OS 1.4, the “PCICC” keyword is introduced in the RACDCERT GENCERT function. When “PCICC” is specified, a PCICC (on 9672 or z900/z800), PCIXCC, or Crypto Express2 (on z990/z890) cryptographic coprocessor is used for generation of the key pair.

Notes:

- The private key generated with the PCICC keyword is stored in the PKDS.
- The PCICC keyword is required to generate keys with the current RACF maximum key length of 2048 bits.
- If there is no hardware cryptographic coprocessor to provide the key generation service, there is no alternate key generation by software. That is, once the PCICC keyword is specified, it must be a coprocessor available to perform the key pair generation or the operation fails.

Important: RSA key lengths in RACF:

- 1024 bits is the maximum key length that is not export controlled, as of the writing of this book.

  Key lengths between 512 and 1024 bits are generated in RACF either by software or hardware, depending on whether the keyword PCICC is used.
- RACF can generate 2048 bit RSA keys; however, as this length is export controlled, the generation is tied to the availability of the hardware crypto coprocessor on the system.

Still for the same reason a user cannot RACDCERT ADD a 2048-bit key to RACF.

2.5.5 Certificate and key export formats

In this section we provide updates on the format of the certificates and keys that can be exported from the RACF database or added to the RACF database. The complete syntax of the RACDCERT ADD and RACDCERT EXPORT functions are shown in Figure 2-36 and Figure 2-37 on page 52 below. The keywords pertaining to the discussion in this section are shown in bold characters.

```
RACDCERT [ ID(userid) | MULTIID | SITE | CERTAUTH ]
ADD(data-set-name) [ TRUST| NOTRUST | HIGHTRUST ] [ WITHLABEL('label-name') ]
[ PASSWORD('pkcs12-password') ] [ ICSF ]
```

Figure 2-36 The RACDCERT ADD function syntax

```
RACDCERT [ ID(userid) | MULTIID | SITE | CERTAUTH ]
EXPORT (LABEL ('label-name') ) DSN (output-data-set-name)
[FORMAT (CERTDER | CERTB64 | PKCS7DER | PKCS7B64 | PKCS12DER | PKCS12B64) ]
[ PASSWORD ('pkcs12-password') ]
```

Figure 2-37 The RACDCERT EXPORT function syntax

The PKCS#12 format

The PKCS#12 file format is used to securely transport private keys and associated certificates by encrypting them with a file password. The PKCS#12 specifications are provided at:

http://www.rsasecurity.com/rsalabs/pkcs

RACF is supporting PKCS#12 Version 3 for exporting or adding keys and certificates from and into the RACF database.

The authority to export certificates and private keys in PKCS#12 format is provided by allowing the user to issue the RACDCERT command EXPORT with the PASSWORD sub-keyword. The user *must* have one of the following authorities:

- RACF SPECIAL
- READ permission to FACILITY Class profile IRR.DIGTCERT.EXPORTKEY to EXPORT one's own certificate and private key.
- CONTROL permission to FACILITY Class profile IRR.DIGTCERT.EXPORTKEY to EXPORT another user's certificate and private key, or a SITE or CERTAUTH certificate and private key

The RACDCERT ADD function supports one or more X.509 certificates and private keys contained within a PKCS#12 DER encoding package. PKCS#12 is also known as Private Information Exchange (PFX). The obsolete PFX V0.02 standard is not supported.

As of z/OS 1.6, RACF adds the complete certificate chain from the PKCS#12 package, whereas on previous releases it was limited to the first certificate found in the package. In order to be authorized to RACDCERT ADD keys and certificates contained in a PKCS#12 file, the user must either:

- Be RACF SPECIAL.
- Have READ permission to FACILITY Class profile IRR.DIGTCERT.ADD to perform the function for himself.
- Have UPDATE permission to FACILITY Class profile IRR.DIGTCERT.ADD to perform the function for others.
- Have CONTROL permission to FACILITY Class profile IRR.DIGTCERT.ADD to perform the function for SITE and CERTAUTH.

Attention: If the user is not SPECIAL and does not have CONTROL permission to IRR.DIGTCERT.ADD in the FACILITY class, then only end-entity certificates will be added from the PKCS#12 package into RACF.

The PKCS#7 format

The PKIX standards account for certificates hierarchy, where an end-entity certificate (for example, a server certificate) is signed by a chain of signing certificate, as shown in

Figure 2-38 on page 53. At the top of this certificate hierarchy there is the *root*, or self-signed, certificate.

A complete certificate chain can be exported or added from and into the RACF database using the PKCS#7 certificate package file.

- RACDCERT ADD adds the complete chain of certificates, and the certificates are added in the order of top CA down to subject (end-entity). The CA certificates are added as CERTAUTH if the user is authorized; that is, if the user has CONTROL permission to FACILITY class profile IRR.DIGTCERT.ADD or is RACF SPECIAL.

  If not specified on the command, RACDCERT will determine the TRUST status of the top CA dynamically:

  – TRUST if self-signed or the signer is already trusted; NOTRUST otherwise
  – TRUST of subsequent certificates inherited from signer
  – Inconsistencies cause NOTRUST to be set for the examined certificate (expired certificate, unknown signature algorithm, etc.)

- RACDCERT EXPORT can locate CERTAUTH certificates to build a complete PKCS#7 chain. The FORMAT keyword must indicate a PKCS#7 value:

  – PKCS7DER for a DER (binary) encoded PKCS7 chain
  – PKCS7B64 for the same DER encoding followed by an additional base64 encoding

  The user must have CONTROL permission to FACILITY class profile IRR.DIGTCERT.EXPORT or be RACF SPECIAL.

  RACF issues a informational message if the constructed chain is incomplete.

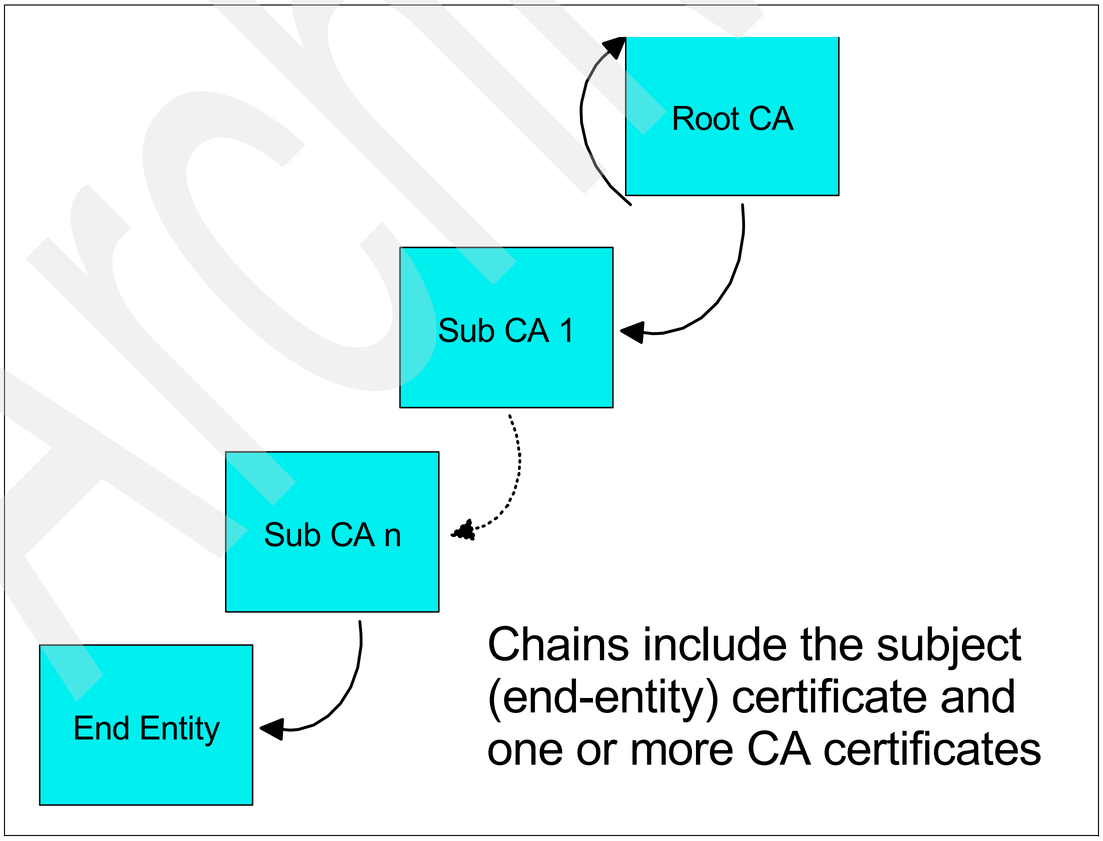

Root CA
Sub CA 1
Sub CA n
End Entity
Chains include the subject (end-entity) certificate and one or more CA certificates

Figure 2-38 A certificate chain

Example of handling a PKCS#7 chain of certificates with RACDCERT command

In this section we give an example of handling a PKCS#7 chain of certificates with the RACDCERT command.

Building the certificates chain

We use the commands shown in Figure 2-39 to build the certificates chain. We end up with a chain of three certificates:

- A self-signed top CA certificate, with a label of 'test root ca'
- An intermediate CA certificate: 'test intermediate ca'
- An end-entity certificate: 'test subject'

Note that all these certificates are created in the RACF database with an associated private key.

```
RACDCERT CERTAUTH GENCERT SUBJECTSDN(CN('TEST ROOT CA')        +
   O('PSSCRACF')                                               +
   C('FR'))                                                    +
    WITHLABEL('test root ca')
RACDCERT CERTAUTH GENCERT SUBJECTSDN(CN('test intermediate ca') +
   O('MOPPSSC') C('FR'))                                       +
   WITHLABEL('test intermediate ca')                           +
   KEYUSAGE (CERTSIGN)                                         +
   SIGNWITH(CERTAUTH LABEL('test root ca'))
RACDCERT ID(PATKAP) GENCERT SUBJECTSDN(CN('test subject')      +
   O('MOPPSSC') C('FR'))                                       +
   WITHLABEL('test subject')                                   +
   SIGNWITH(CERTAUTH LABEL('test intermediate ca'))
```

Figure 2-39 Creating the certificates chain

Creating a PKCS#7 file

We use the command shown in Figure 2-40 to export this certificate chain as a PKCS#7 file.

```
RACDCERT ID(PATKAP) EXPORT(LABEL('test subject'))              +
   DSN('PATKAP.PKCS7.CHAIN')                                   +
   FORMAT(PKCS7B64)
```

Figure 2-40 Creating the PKCS#7 file

Note that only the certificate is exported here. The associated private key, as opposed to what happens with the PKCS#12 standard, is not.

Adding the PKCS#7 chain to the RACF database

In this section we delete the original certificates to re-introduce them from the PKCS#7 file. The PKCS#7 chain is added using the command shown in Figure 2-41.

```
RACDCERT ID(PATKAP)                                             +
   ADD('PATKAP.PKCS7.CHAIN')                                    +
   WITHLABEL('test subject')
```

Figure 2-41 Adding the PKCS#7 chain

We proceeded here with three different setups to add the chain of certificates, which we explain in the next sections.

1. The initial ‘test subject’ certificate has been deleted in the RACF database before the RACDCERT ADD operation.

   When performing the RACDCERT ADD function, RACF leaves the CAs certificates (that is, ‘test intermediate ca’ and ‘test root ca’) unchanged in the RACF database and fetches only the end-entity certificate from the PKCS#7 file and stores it with the label ‘test label’. Note that there is no private key associated with this added certificate. The certificate’s content is shown in Figure 2-42.

```
Label: test subject
Certificate ID: 2QbXwePSwdejhaKjQKKkgpGFg6NA
Status: TRUST
Start Date: 2005/05/25 00:00:00
End Date:   2006/05/25 23:59:59
Serial Number:
     >01<
Issuer's Name:
     >CN=test intermediate ca.O=MOPPSSC.C=FR<
Subject's Name:
     >CN=test subject.O=MOPPSSC.C=FR<
Private Key Type: None
Ring Associations:
*** No rings associated ***
```

Figure 2-42 The added end-entity certificate

2. The initial ‘test subject’ and ‘test intermediate ca’ certificates have been deleted in the RACF database before performing the RACDCERT ADD operation.

   This time RACF leaves the ‘test root ca’ certificate unchanged in the database. The ‘test subject’ certificate is added, and so is the intermediate CA certificate, which is added as CERTAUTH. However, the intermediate CA certificate gets a RACF specified label: ‘LABEL00000001’. The certificate is shown in Figure 2-43.

```
Label: LABEL00000001
Certificate ID: 2QiJmZmDhZmjgdPBwsXT8PDw8PDw8PFA
Status: TRUST
Start Date: 2005/05/25 00:00:00
End Date:   2006/05/25 23:59:59
Serial Number:
     >01<
Issuer's Name:
     >CN=TEST ROOT CA.O=PSSCRACF.C=FR<
Subject's Name:
     >CN=test intermediate ca.O=MOPPSSC.C=FR<
Key Usage: CERTSIGN
Private Key Type: None
Ring Associations:
*** No rings associated ***
```

Figure 2-43 The added intermediate CA certificate

3. All three initial certificates have been deleted in the RACF database prior to performing the RACDCERT ADD operation. All three certificates are fetched by RACF from the PKCS#7 file and installed in the database. The certificates next to the end-entity

certificates (that is, the intermediate and root CA certificates) are given a label by RACF: ‘LABEL00000001’ and ‘LABEL00000002’, and are installed as CERTAUTH. The root CA certificate added from the PKCS#7 file is shown in Figure 2-44 on page 56.

```
Label: LABEL00000001
Certificate ID: 2QiJmZmDhZmjgdPBwsXT8PDw8PDw8PFA
Status: TRUST
Start Date: 2005/05/25 00:00:00
End Date:   2006/05/25 23:59:59
Serial Number:
     >00<
Issuer's Name:
     >CN=TEST ROOT CA.O=PSSCRACF.C=FR<
Subject's Name:
     >CN=TEST ROOT CA.O=PSSCRACF.C=FR<
Key Usage: CERTSIGN
Private Key Type: None
Ring Associations:
*** No rings associated ***
```

Figure 2-44 The added root CA certificate

Note that in all cases the TRUST status is dynamically inherited from the root CA certificate. In the last case, when the root CA certificate has to be fetched from the PKCS#7 file, it is given a TRUST status by RACF, as it is a self-signed certificate.

2.5.6 New default CA certificates in the RACF database

Since z/OS 1.4, the following changes have been made to the set of default certificates installed in the RACF database at IPL time:

- The IBM World Registry™ CA has been removed.
- The following expiring certificates have been replaced:
  - Verisign Class 2 Primary CA
  - Verisign Class 3 Primary CA
- The following certificates have been added:
  - Verisign Class 1 Individual CA
  - Verisign Class 2 Individual CA
  - Verisign International Svr CA
  - Identrus Interoperability CA - Self-signed root
  - GTE CyberTrust Root CA - Self-signed root
  - Entrust.net Secure Server CA - Signed by GTE root

2.5.7 RACF certificate management enhancements at z/OS 1.6

Two new functions are introduced: The RACDCERT REKEY and the RACDCERT ROLLOVER. The idea behind these functions’ implementation is to make the RACDCERT functions compliant with the Identrus standards.

The following is a quote of the Identrus organization at http://www.identrus.com:

> Identrus has built a regulated policy framework that provides financial institutions (FIs) and their clients with the global standard for digital identity authentication. Identrus identity credentials are based on a comprehensive legal and technical framework that creates an FI-centric trust community. Identrus identity credentials are flexible and interoperable, supporting enterprise-level applications as well as domestic and global digital transactions.

In an Identrus system, private keys do not outlive their certificates. If a certificate is to be reissued, a new key pair is generated, and certificates created prior to the expiration of the old certificate. This is called key rollover, or certificate rekey. Note that both certificates would be valid for some period of time as the old certificate is still usable until it expires.

In RACF, key rollover is performed by using the RACDCERT REKEY and ROLLOVER operands:

- RACDCERT REKEY makes a self-signed copy of the original certificate, with a new public-private key pair.
- RACDCERT ROLLOVER finalizes the rekey operation by:
  - Replacing or supplementing the original certificate with the new certificate in every keyring the original certificate is connected to
  - Destroying the original private key
  - Copying over the serial number basing information in case the certificate is being used to sign other certificates (The serial number basing information is an internal counter kept in the certificate profile and used to generate issuer's serial numbers in certificates this certificate signs.)

2.5.8 The RACDCERT REKEY function

The syntax of the command is shown in Figure 2-45.

```
RACDCERT [ ID(userid) | SITE | CERTAUTH ]
REKEY(LABEL('existing-label-name') ) [ SIZE(key-size)] [ NOTBEFORE([DATE (yyyy-mm-dd) ]
[ TIME(hh:mm:ss) ] ) ] [ NOTAFTER([DATE(yyyy-mm-dd) ] [ TIME(hh:mm:ss)]) ]
[ ICSF | PCICC ] [ WITHLABEL('to-be-created-label-name') ]
```

Figure 2-45 The RACDCERT REKEY function

The following parts of the original certificate are copied to the replicated certificate:

- Subject's distinguished name
- Key usage
- Subject alternate name

The replicated certificate is then self-signed and stored as a new certificate profile associated with the same user ID, CERTAUTH, or SITE. The original certificate is unaltered by this operation.

Example of RACDCERT REKEY operation

For this example we use the initial end-entity certificate, 'test subject', that we were using in the previous PKCS#7 example, as REKEY and ROLLOVER apply to certificates with an associated private key. We also take this opportunity to connect it to a RACF keyring to further demonstrate what happens with the ROLLOVER function. The RACF commands that we use are shown in Figure 2-46 on page 58, and the resulting certificate is shown in Figure 2-47 on page 58.

```
RACDCERT ID(PATKAP)                                                 +
   ADDRING(rekey_test_ring)
RACDCERT ID(PATKAP)                                                 +
   CONNECT(ID(PATKAP) LABEL('test subject')                         +
   RING(rekey_test_ring))
RACDCERT ID(PATKAP)                                                 +
   REKEY(LABEL('test subject'))                                     +
   WITHLABEL('rekeyed test subject')
```

Figure 2-46 REKEY setup and invocation

```
Label: rekeyed test subject
Certificate ID: 2QbXwePSwdeZhZKFqIWEQKOFoqNAoqSCkYWDoOBA
Status: TRUST
Start Date: 2005/05/25 00:00:00
End Date:   2006/05/25 23:59:59
Serial Number:
     >02<
Issuer's Name:
     >CN=test subject.O=MOPPSSC.C=FR<
Subject's Name:
     >CN=test subject.O=MOPPSSC.C=FR<
Private Key Type: Non-ICSF
Private Key Size: 1024
Ring Associations:
*** No rings associated ***
```

Figure 2-47 The rekeyed end-entity certificate

If the replicated certificate needs to be signed by another certificate in RACF or another certificate authority, a PKCS#10 request may be created from it using the RACDCERT GENREQ function. This request may be input to RACDCERT GENCERT to sign with another certificate in RACF or sent to the external certificate authority for fulfillment. This would need to be performed before the rekeying is finalized with the ROLLOVER function.

We actually managed to get a certificate signed by the root CA using the commands shown in Figure 2-48. The resulting signed certificate is shown in Figure 2-49 on page 59.

```
RACDCERT ID(PATKAP)                                                 +
   GENREQ(LABEL('rekeyed test subject'))                            +
   DSN('PATKAP.REKEY.CERT')
RACDCERT ID(PATKAP)                                                 +
   GENCERT('PATKAP.REKEY.CERT')                                     +
   WITHLABEL('signed rekeyed test subject')                         +
   SIGNWITH(CERTAUTH LABEL('test root ca'))
```

Figure 2-48 Generating the 'signed rekeyed test subject' certificate

```
Label: signed rekeyed test subject
Certificate ID: 2QbXwePSwdeiiYeVhYRAmYWShaiFhECjhaKjQKKkgpGFg6NA
Status: TRUST
Start Date: 2005/05/25 00:00:00
End Date:   2006/05/25 23:59:59
Serial Number:
     >05<
Issuer's Name:
     >CN=TEST ROOT CA.O=PSSCRACF.C=FR<
Subject's Name:
     >CN=test subject.O=MOPPSSC.C=FR<
Private Key Type: Non-ICSF
Private Key Size: 1024
Ring Associations:
*** No rings associated ***
```

Figure 2-49 The signed rekeyed certificate

2.5.9 The RACDCERT ROLLOVER function

The syntax of the command is shown in Figure 2-50.

```
RACDCERT [ ID(userid) | SITE | CERTAUTH ]
ROLLOVER(LABEL('old-label-name') ) NEWLABEL('new-label-name') [ FORCE ]
```

Figure 2-50 The RACDCERT ROLLOVER function

Rollover performs the following three actions in order:

1. Deletes the private key of the source certificate so that it may not be used again for signing or encryption.
2. Connects the target certificate to every keyring the source certificate is connected to. If the source certificate is connected with USAGE(PERSONAL) then the source certificate is removed from the keyring.
3. Copies the serial number base from the source certificate to the target certificate.

Once rollover is complete, the new certificate may be used as though it were the old certificate. The old certificate is retained for historical reasons (such as validating signatures on existing certificates), but may no longer be used for any private key operations such as signing other certificates.

The FORCE option specifies that RACF should bypass some error checking and perform the operation unconditionally. Without specifying the FORCE keyword, the following conditions must all be true; otherwise an error message is issued and the command ends:

1. The values specified for the LABEL and NEWLABEL keywords must be different.
2. The certificates identified by the LABEL and NEWLABEL keywords must both have private keys associated with them.
3. The certificate identified by the NEWLABEL keyword must have never been the target of a previously issued RACDCERT ROLLOVER function and never been used to sign other certificates.

When the FORCE keyword is specified, the checks listed above are not performed.

Note that the ROLLOVER function has a degenerative feature, in which if the same certificate is specified for both the LABEL and NEWLABEL keywords and the FORCE keyword is also specified, the private key of this certificate is deleted.

Example of RACDCERT ROLLOVER operation

We resume where we were at the completion of the REKEY example; that is:

- The initial end-entity certificate ‘test subject’ is connected to the keyring ‘rekey_test_ring’.
- A rekeyed certificate for this end-entity certificate has been prepared as the ‘signed rekeyed test subject’.

We now proceed with the ROLLOVER itself by issuing the command shown in Figure 2-51.

```
RACDCERT ID(PATKAP)                                              +
   ROLLOVER(LABEL('test subject'))                               +
   NEWLABEL('signed rekeyed test subject')
```

Figure 2-51 The RACDCERT ROLLOVER command

Again displaying the certificates connected to the ‘rekey_test_ring’ yields the result shown in Figure 2-52.

```
 Ring:
      >rekey_test_ring<
 Certificate Label Name             Cert Owner     USAGE      DEFAULT
 --------------------------------   ------------   --------   -------
 signed rekeyed test subject        ID(PATKAP)     PERSONAL     NO
```

Figure 2-52 The replacement of a connected certificate with the ROLLOVER function

Now, looking again at the content of the initial end-entity certificate, we see in that the certificate no longer has an associated private key.

```
Label: test subject
Certificate ID: 2QbXwePSwdejhaKjQKKkgpGFg6NA
Status: TRUST
Start Date: 2005/05/25 00:00:00
End Date:   2006/05/25 23:59:59
Serial Number:
     >01<
Issuer's Name:
     >CN=test intermediate ca.O=MOPPSSC.C=FR<
Subject's Name:
     >CN=test subject.O=MOPPSSC.C=FR<
Private Key Type: None
Ring Associations:
*** No rings associated ***
```

Figure 2-53 The initial certificate no longer has an associated private key

3

Multilevel Security and RACF

Multilevel Security (MLS) is not new to RACF, as the base features of MLS have been implemented in RACF since 1990. However, MLS was not addressing at this time specific resources such as the TCP/IP resources or the Unix System Services resources. This is addressed in enhancements to RACF MLS at z/OS 1.5 and z/OS 1.6. In this chapter we introduce the principles of operation of an MLS system and how they are implemented in RACF.

3.1 An introduction to MLS

First, in this section, we provide an introduction to MLS.

3.1.1 What is Multilevel Security

A Multilevel Security system is a security environment that allows the protection of data based on both traditional Discretionary Access Controls, and controls that check the sensitivity of the data itself through Mandatory Access Controls.

These Mandatory Access Controls are at the heart of a Multilevel Security environment, which prevents unauthorized users from accessing information at a classification they are not authorized to, or changing the classification of information they do have access to. These Mandatory Access Controls provide a way to segregate users and their data from other users and their data regardless of the discretionary access they are given though access lists, etc.

Creating a Multilevel Security environment requires a combination of several software and hardware components that enforce the security requirements needed for such a system. The security-relevant portion of software and hardware components that make up this system can also be known as the trusted computing base.

3.1.2 Why Multilevel Security

The primary arena where Multilevel Security is valuable is government agencies that need a security environment that keeps information classified and compartmentalized between users. In addition to the fundamental identification and authentication of users, auditing and accountability of the actions by authenticated users on these systems is provided by the security environment.

Normally in such highly secure environments, to manage the compartmentalization of information between users, each compartment is on its own system, making it difficult for classified information to spill from one system to another, since the connections between systems can be highly controlled. With Multilevel Security, these systems can be consolidated onto a single system, with each compartment independent of the other, so that no transfer of data can occur between compartments within that system. This will take advantage of the cost savings of not having to manage multiple systems, but only a few, or one, system.

Commercial clients may also find some features of Multilevel Security useful, such as to separate sensitive client information from the general populace, or from another user. New government regulations, such as HIPAA, or corporate mergers, are examples where security of information based on the information itself is important in the commercial world.

RACF, also known as the z/OS Security Server, has several options that can be turned on and off to manipulate different aspects of a Multilevel Security environment. Because it is possible to have some features of Multilevel Security on at one time (creating a partial Multilevel Security environment), commercial clients may find this type of environment useful to meet these needs vs. running a full fledged Multilevel Security environment.

3.1.3 Access controls

Discretionary Access Control (DAC) allows access to data to be controlled by the discretion of the owner of the data (or those with administrative authority to the data). Since access could be given out at the discretion of an authorized person to the data, it would be quite easy to unintentionally give access to data to people who do not need access to it. For example, the owner of some data gives access to a GROUP since that group of people needs access to his data, then the group owner connects a new user to the GROUP since the new user needs access to resources that the GROUP has access to, not knowing that the GROUP also has access to sensitive data that this new user should not have access to.

Mandatory Access Control (MAC) imposes additional restrictions upon users so they now access data based on a comparison of the classification of the user and the classification of the data as well as the standard DAC checking. This additional security check verifies that users can only access data and resources that their classification allows them to, even though their discretionary access will allow them to access such data or resources.

Mandatory Access Control was implemented in RACF 1.9 with MVS/ESA 3.1.3 (in 1990) as part of the effort to meet the US Government criteria (TCSEC B1) as specified in the *Department of Defense Trusted Computer System Evaluation Criteria,* DoD 5200.28-STD (also know as the TCSEC or Orange Book).

In addition to checking a user's access to data based on the classification of the user and the data, the z/OS trusted computing base implementation of Multilevel Security provides some of these additional functions as well:

- The system does not allow a storage object to be reused until it is purged of residual data.
- The system enforces accountability by requiring each user to be identified, and creating audit records that associate security-relevant events with the users who cause them.
- The system labels all hardcopy with security information.
- The system optionally hides the names of data sets, files, and directories from users who do not have access to those data objects.
- The system does not allow a user to declassify data by "writing down" (that is, writing data to a lower classification than the classification at which it was read) except with explicit authorization to do so.
- The system does not allow a user to view rows of data in a DB2® table that he is not classified to view.

3.1.4 Introduction to Mandatory Access Control

There are several principles one needs to comprehend to fully understand the interactions of MAC checking—the given classification for a particular piece of data, a resource, or a user as defined by the security label for that item. These security labels are then used to determine what access will be allowed; for example, a user looking at some data in a file, the subject's (in this case the user) security label is compared to the object's (in this case the file) security label. A dominance or equivalence relationship is determined, should one exist, and the type of access requested by the subject onto the object is determined; and based on that, MAC access to the object will be determined.

Breaking this down, one can see that there are three major principles that should be understood. They are:

- Security labels
- The type of relationship between security labels (dominance, equivalence, and disjoint)
- The type of access requested (MAC access)

Security labels

A security label is a combination of a hierarchical level of classification (security level) and a set of zero or more non-hierarchical categories (security category).

In Figure 3-1 we show a basic table that has a list of hierarchical levels from 1 to 4, where 1 is the lowest and 4 is the highest security level, and a list of non-hierarchical categories from category A to category E. Using this table we can quickly create several security labels combining a security level, and zero or more categories. Some example security labels are:

- L1A is a combination of security level 1 with category A.
- L2BCD is a combination of security level 2 with categories B, C, and D.
- L3N is a combination of security level 3 with no categories.

Note: The security labels we use for this example are defined as they are for ease of use, and to allow the reader to quickly determine a security label's security level and categories. So as an example, using Figure 3-1, one could also create the additional security labels:

- ARTHUR is a combination of security level 2 with category C.
- FORD is a combination of security level 3 with categories A, C, and E.
- MARVIN is a combination of security level 3 with no categories. This security label is also equivalent to L3N.

The table below has several more security labels defined, and one could easily create several more based on this simple example.

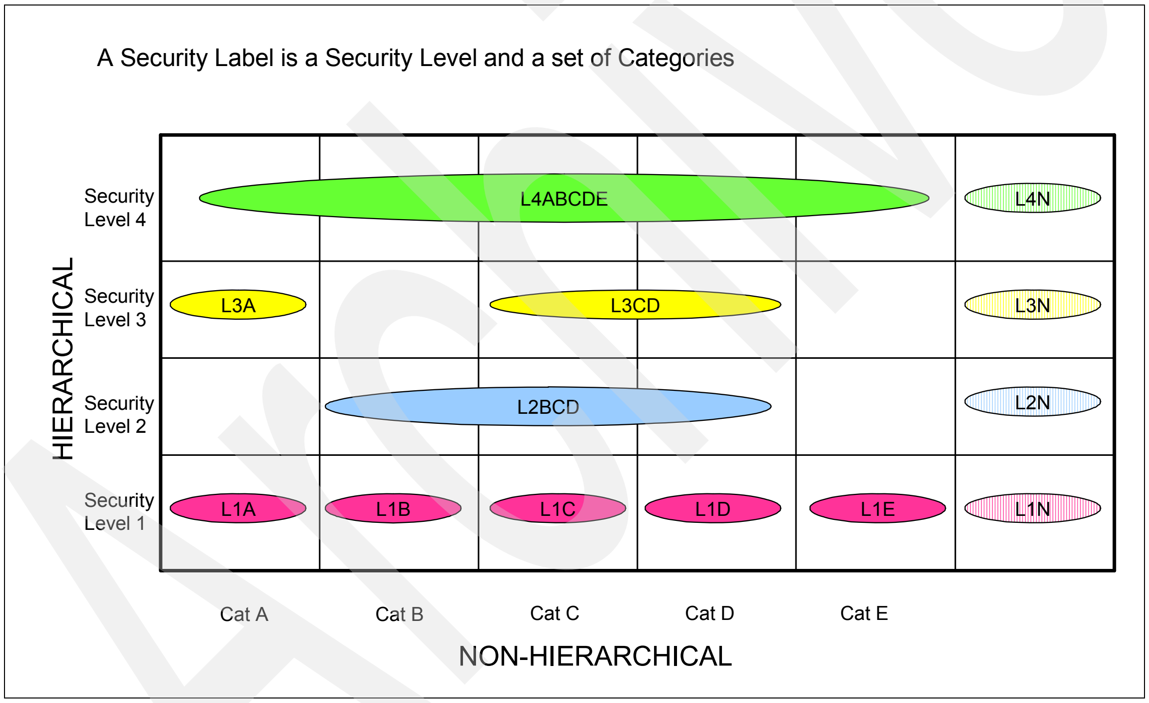


Figure 3-1 Simple security label diagram

Dominance, equivalence, and disjoint

Access is granted or rejected based on the relationship between the subject's security label and the object's security label, and the type of access that is requested. Two important relationships between security labels are dominance and equivalence. Though other relationships between two security labels may exist, in those cases authority to the object is not allowed.

Dominance

For security label A to dominate security label B the following must be true:

- The security level of A is greater than or equal to the security level of B.
- Security label A has at least all the categories that define security label B.

Returning to Figure 3-1 on page 64, security label L3CD, for example, would dominate security labels L1N, L1C, L1D, L2N, L3N, and L3CD, since L3CD is at an equivalent or higher security level then the security labels it is dominating, and it contains all the categories that these security labels have. Notice that even though security label L2N has no categories, it will be dominated by all security labels at a higher or equivalent security level and any categories.

With reverse dominance access checking, the access rules are the reverse of the access rules for dominance access checking.

Equivalence

For security label A to be equivalent to security label B, the following must be true:

- The security level of A is equal to the security level of B.
- Both security labels A and B must have the same set of categories.

Another way one can say that two security labels are equivalent is if both security labels dominate each other.

Returning to Figure 3-1 on page 64, the only case of equivalence in the figure is if a security label is being compared to itself (that is, the subject and object security labels are the same).

Disjoint

If neither security label dominates the other then the two security labels are said to be *disjoint* or *incompatible.*

MAC access

In addition to the type of relationship, as documented above, the type of access that is requested is also important. There are only three types of accesses that can be requested: Read-only, read/write, and write-only.

Read-only test

The users intend to only read information; therefore, they should be able to read information at their classification or information at a lower classification. In terms of MAC checking, the users' security label must *dominate* the object's security label that they intend to read. This allows them to read information covered by their security label, but not information of a higher level, or outside the users' category.

Write-only test

The user intends to only write information; therefore, he should be able to write information at his classification or information at a higher classification. In terms of MAC checking, the object's security label that he intends to write to must *dominate* the user's security label. This allows him to write information at his security label or higher, but not information of a lower level, or outside the user's category (that is, he will not be able to declassify information).

Read/write test

The user intends to read and write information. Like in the read-only case, the user's security label must be able to *dominate* the object's security label; *and*, like the write-only case, the object's security label must be able to *dominate* the user's security label. Therefore, for the user to do a read/write action to the object, the user's and object's security labels must both be *equivalent* to each other. This allows his to read and write information aboutly at his security label, but not outside his security label (that is, he will not be able to declassify information).

Controlled write-down

There might be cases where you want to allow for controlled situations of write-down. The security administrator can assign a *write-down by user* privilege to individual users or groups of users that allows them to select the ability to write down. The security administrator activates and deactivates the privilege by creating the profile IRR.WRITEDOWN.BYUSER in the facility class. A user can activate write-down mode if the profile exists, the user has at least read access to it, and the FACILITY class is active and SETROPTS RACLISTed. If the user has update or higher access to the profile, write-down mode is active by default when the user enters the system. RACF provides the RACPRIV command and z/OS UNIX provides the **`writedown`** command, which allows users who are authorized to the write-down privilege to reset and query the setting of there write-down mode.

Access rules

Access rules depend on the purpose for which a subject accesses an object and whether the subject is allowed to write down.

The subject is not allowed to write down

A subject is not allowed to write down if the RACF MLS option is active and the security administrator has not set up controlled write-down and given the subject authorization to write down. This should be the case for most users in a multilevel secure environment. In this case the access rules, depending on the purpose for which the subject accesses an object, are:

- Read only: A subject can read an object when the subject's security label dominates the object's security label.
- Write only: A subject can write to an object when the object's security label dominates the subject's security label.
- Read/write: A subject can read from and write to an object only if the security labels of the subject and object are equivalent.

Only the basic case is documented here—that is, a normal MAC check is being done with the *no write-down* rule in effect. With RACF there are several other RACF options that could affect how these three different MAC access tests are done, which are covered in more detail in Chapter 2, "Security labels," of *Multilevel Security and DB2 Row-Level Security Revealed*, SG24-6480.

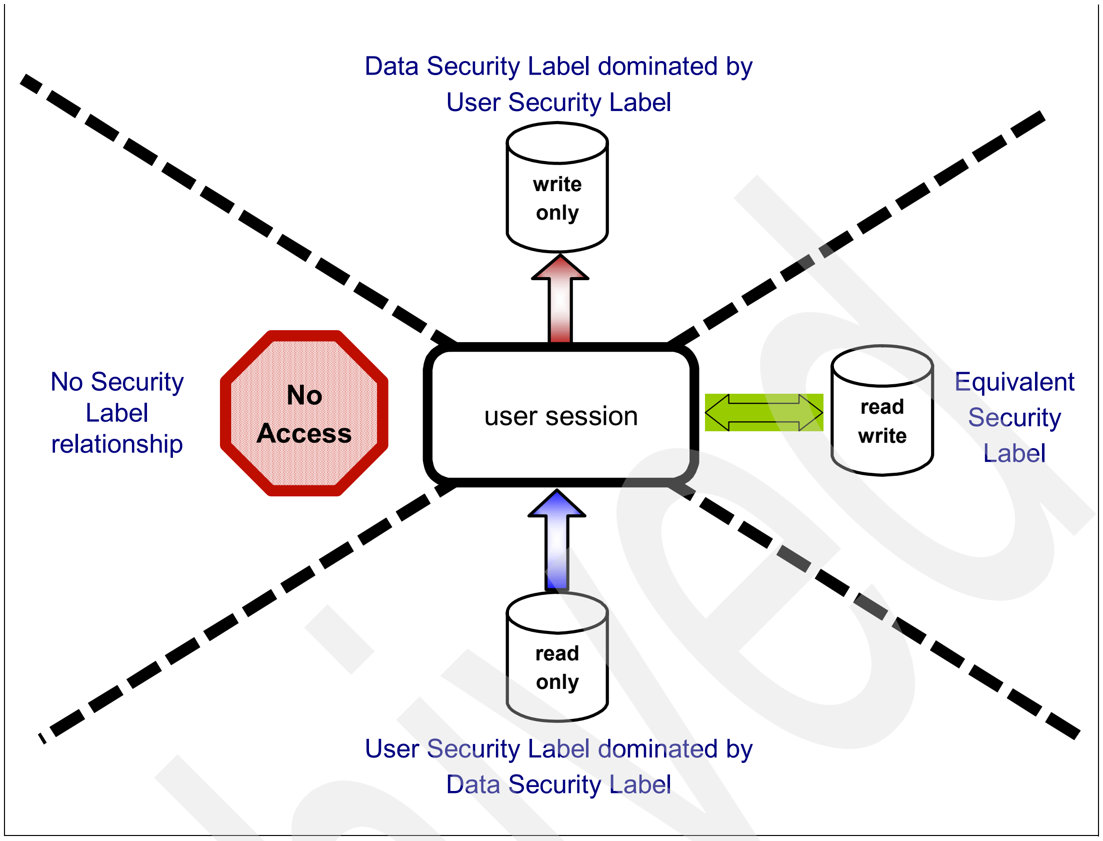

Data Security Label dominated by User Security Label
write only
No Security Label relationship
No Access
user session
read write
Equivalent Security Label
read only
User Security Label dominated by Data Security Label

Figure 3-2 MAC principle

To summarize the basic MAC principle (as seen in Figure 1-2):

- Should the user's security label dominate the data's security label, the user can read the data.
- Should the data's security label dominate the user's security label, the user can write to the data.
- Should the user's and data's security labels be equivalent, then the user can read and write to the data.
- Should there be no equivalence or dominance relationship between the user's security label and data's security label, the user will be denied access to the data.

3.2 Multilevel Security in z/OS with RACF

Even though the theoretical implementation of Multilevel Security may seem reasonable, the actual implementation that is done by a trusted computing base may be slightly different. The trusted computing base, in this case z/OS, must make certain decisions as to what defines certain actions, then make the appropriate security decision based on that definition of that action. This is important to understand because what one might think is a certain type of MAC access, the trusted computing base may have decided is a different type of MAC access.

For example, in z/OS the case of a user attempting to do a read-only or read/write action against a data set is straightforward enough. But because of the way z/OS handles data set processing, there is no real case, in the z/OS trusted computing base, where one could do a write-only action against a data set. This is because every time a user needs to write to a data set, z/OS must read some part of the data set before preforming the write action. As a result, the z/OS trusted computing base has decided that there is no case where a it will do a write-only test against a data set, so no such test exists.

So as one can see from this simple example, it is not only important to know how Multilevel Security works, but also how it is implemented by the trusted computing base.

3.2.1 SECLABELs

In RACF, security labels are defined in the SECLABEL class, and are often called SECLABELs (vs. security labels). As discussed earlier, a SECLABEL is a combination of security level (saved in the SECLEVEL profile in the SECDATA class), and set of zero or more categories (saved in the CATEGORY profile in the SECDATA class).

Users are then granted access to any SECLABELs they need authority to. Although a user can only have one SECLABEL active at a time for a session, at logon time, he can request to log on with any SECLABEL he has authority to; so during the life of that session, he is considered to be at the SECLABEL he logged on with. Resources, such as a data sets, that require a security label are given SECLABELs as well.

Now with all the SECLABELs in place and the SECLABEL class activated, when a user requests access to resources (such as a user reading a data set), a MAC check is done using the user and resource SECLABELs, followed by a DAC check using the resource's access list. Note that MAC will occur first (once the SECLABEL class is activated), then DAC.

3.2.2 Multilevel Security in action

Now that we have a basic idea of how Multilevel Security works, let us go though a simple example of Multilevel Security in action.

In the following example, we use the security labels that were defined in Figure 3-1 on page 64. We have the user MATT and provide him with the authority to the SECLABEL L4ABCDE (that is, he has a security level of 4, and access to categories A, B, C, D, and E), and all the data sets he owns have his SECLABEL. The user ROLAND has authority to the SECLABEL L3CD, and all the data sets he owns have his SECLABEL. Finally, the user CHRIS has authority to the SECLABEL L1A, and all the data sets he owns have his SECLABEL as well. In addition, we have data sets defined to the group ITSO, and give all the data sets with its high level qualifier a SECLABEL of L1D. Let us finally assume that all the users also have UPDATE authority to all the data sets in this example.

Working with the user ROLAND (who will be the subject taking these actions), he will be able to read and write to his own data sets (the data sets being the object receiving the actions in all these cases) since his user ID will have an equivalent SECLABEL to the data set SECLABEL of L3CD.

The user ROLAND can now only read those data sets that belong to the group ITSO, since ROLAND's SECLABEL of L3CD will dominate the ITSO's data set SECLABEL of L1D (that is, his SECLABEL's security level is greater then the ITSO's security level, and his SECLABEL contains all the categories that the ITSO's SECLABEL has).

He cannot read anything in the data sets belonging to MATT since the security level is higher then his, and his SECLABEL has more categories then ROLAND's SECLABEL (that is, the MAC check fails), even though his discretionary access allows him access to those data sets.

Also, ROLAND cannot access any of the data sets that belong to CHRIS. Even though ROLAND's SECLABEL has a higher security level then CHRIS's, the categories that

ROLAND’s SECLABEL can look at are not included in the categories that CHRIS has access to.

The above is a very simple picture of Multilevel Security in a z/OS environment. One can also see that movement of data is very restrictive. It is also possible to implement only certain parts of Multilevel Security, to create a partial Multilevel Security environment. This reduces some of the restrictions on the movement of data, though making a less secure system (from an Multilevel Security point of view).

3.2.3 DB2 and Multilevel Security

With the introduction of DB2 V8, DB2 has taken active participation in Multilevel Security as well. DB2 has provided row-level support so that rows can be protected based on the SECLABEL of the row. This allows multiple users to access a particular table, yet allows only those users with the correct SECLABEL to read or update the particular rows they have access to.

Additionally, with the RACF Access Control Module installed, SECLABELs can be assigned to tables or views, for example, so that only those users with the correct SECLABEL can read or update the table or view they have access to.

3.2.4 Before turning on Multilevel Security

A very important part of preparing to run in a Multilevel Security environment, or even a partial Multilevel Security environment, is planning. There is no way to emphasize this more than to say that the next most important thing is planning.

To simply put it, *planning is key*.

Part of the complexity that is created by this environment is the approach that we have seen most people take towards Multilevel Security. Commonly, it is taking a system and breaking it down into multiple compartmental environments, though in truth it maybe better to take an approach of imagining things as individual compartmentalized system being incorporated into a single system. Each compartment is independent of the other, such that no data movement can occur between compartments within that system, yet taking advantage of the fact that everything is on a single system so that individuals that have the security label that allows them to see information at their classification or lower can, without having to move from one compartment to another.

Another part of the complexity of establishing a working Multilevel Security environment (or even a partial environment) is that when turning on or off any feature of Multilevel Security, one is not just turning it on for an application, but for the entire MVS image. Even though there are several Multilevel Security options, these options affect the entire MVS image, not just a single application. So one does not only have to consider the one application that he is using to take advantage of Multilevel Security, but how that will effect the entire MVS image.

So one can see, before starting to work with a Multilevel Security environment (even a partial Multilevel Security environment), proper prior planning will help prevent problems as one starts to establish this in a production environment.

3.3 Multilevel Security vocabulary

As one works with Multilevel Security, there is a lot of vocabulary one needs to be very careful about. Confusion can easily arise while working in this environment, especially since there are several terms that sound very close to one another, or have different meanings in different contexts.

A very good example of this is the term MLS. One use of this term means the RACF option MLS, which activates the *no-write down* option. The term MLS has also been used as a shortcut for the term Multilevel Security. If not used in the correct context, one can easily be confused by which meaning the writer or speaker may be intending.

For the purposes of this book, we have maintained the following terms:

- Multilevel Security - Running in an environment with all the Multilevel Security options turned on. This would include protecting all security-relevant resources by RACF with a security label, and full no-write down protection. To be in this environment, one has taken all the applicable steps as documented in Chapter 3 in *Planning for Multilevel Security and the Common Criteria,* GA22-7509-01.

  Multilevel Security is also used when talking about an environment where the potential of having all the Multilevel Security options turned on could exist.
- Partial Multilevel Security - Running in an environment with some of the Multilevel Security options on.
- MLS or SETR MLS - This term will be reserved for the RACF SETROPTS option MLS, which turns on and off the no-write down protection (also known as the *-property).

The following terms are not very confusing when seen in written text, but when spoken can easily be confused. Just something else to watch out for.

- SECLEVEL - A hierarchical designation for data that represents the sensitivity of the information
- SECLABEL - A name that represents the combination of a hierarchical level of classification and a set of non-hierarchical categories

3.4 Common criteria

Standards have been set and evaluations have been (and are being) done to certify the integrity of the z/OS trusted computing base for Multilevel Security support. Agencies outside of IBM have set these standards and evaluated the trusted computing base, verifying the integrity of IBM's solution for Multilevel Security.

Note: To understand more about how to configure for Common Criteria compliance please refer to *Planning for Multilevel Security and the Common Criteria,* GA22-7509.

As a bit of history, the MVS 3.1.3 trusted computing base (which included RACF, JES2, JES3, TSO, VTAM®, DFP, and PSF) were formally evaluated, and received the National Security Agency list of evaluated products with a B1 level of trust. Eventually the Common Criteria and ISO 15408 superseded the older US Government standards described in the Orange Book, and IBM has submitted z/OS for evaluation under the Common Criteria standard.

eServer zSeries running z/OS

z/OS V1.6 has been awarded EAL3 certification. This certification encompasses Controlled Access Protection Profile (CAPP) at EAL3 and Labeled Security Protection Profile (LSPP) at EAL3. IBM is also in evaluation for the EAL4 for z/OZ V1.7 with the RACF feature.

IBM DB2 UDB for z/OS Version 8 has also been submitted for evaluation under the Common Criteria with a conformance claim of EAL3.

PR/SM™ is designed to prevent the flow of information among logical partitions, providing highly secure isolation. This isolation allows images of zSeries supported operating systems, including z/OS, z/VM®, and Linux® for zSeries, to run in different logical partitions on a single zSeries server. To understand the latest certification levels for the above please refer to the Web sites below for up-to-date information.

- `http://www.ibm.com/servers/eserver/zseries/security/ccs_certification.html`
- `http://www.ibm.com/security/standards/st_evaluations.shtml`
- `http://niap.nist.gov/cc-scheme/`

3.5 More on security labels

In this chapter we describe what security labels are and how they are used, and their association to the RACF resource classes SECLABEL, SECDATA, and SECLEVEL; security labels that the system automatically creates for you; defining security labels; and how to authorize users to use security labels. This section is done with acknowledgments to *z/OS Planning for Multilevel Security and the Common Criteria*, GA22-7509.

3.5.1 Security labels and data classification policies

A *security label* enables an installation to classify subjects and objects according to a data classification policy, identify objects to audit based on their classification, and protect objects such that only appropriately classified subjects can access them.

Subjects and objects

In this book a *subject* is an entity that requires access to system resources. These system resources are *objects*.

Examples of subjects are:

- Human users
- Started tasks
- Batch jobs
- z/OS UNIX daemons

Examples of objects are:

- Data sets
- Row within a DB2 table
- Commands
- Terminals
- Printers
- DASD volumes
- Tapes

Subjects are defined to RACF, for example, a user or started task will have a RACF user ID; objects, other than rows in a DB2 table, are also defined to RACF as either a *resource profile* or *data set profile*. The terms *subject* and *user ID* have the same meaning in this book and can be used interchangeably.

In a multi-level secure system, subjects and objects have a security label associated with them. The security label is defined to RACF in the resource class SECLABEL. Rows in a DB2 table have a security label associated with them by means of a special column of the table that contains only the 8-character security label that defines the security classification of each row in that table. A subject's security label determines whether the subject is allowed to access a particular object. An object's security label indicates the sensitivity of that object's data.

A subject is authorized to use a security label by having been permitted READ access to the resource profile in the SECLABEL class in RACF, which defines the particular security label. A TSO user may have a default security label defined to him in RACF if desired. The appropriate security label can then be chosen or allowed to default at logon time. Other subjects such as batch jobs or started tasks should have their security label defined as their default because it is not possible to choose.

Subjects can have authorization to more than one security label, but can use only one security label at any given time. A TSO user gets assigned his chosen security label at logon time, and this remains for the duration of his session. If he needs to switch security labels, then he will need to log off and log back on again to choose another one.

Access to objects (that is, data) in a multilevel secure system is controlled by both the subject and the object having a security label assigned to them. A subject can access information in an object only when the subject's security label entitles the access. Entitlement to access the data depends on the theory of dominance of the subject's security label over the object's security label. For more details on this see "Dominance" on page 77. If the subject's security label does not have enough authority, the subject cannot access the information in the object.

In a non-multi-level secure system, access to objects or data is controlled by Discretionary Access Control (DAC); in a multi-level secure system, access to objects or data is controlled by Mandatory Access Control (MAC).

A security label is used as the basis for Mandatory Access Control decisions. By assigning security labels, the security administrator can ensure that data of a certain classification is protected from access by a user of a lesser security classification. Security labels provide the capability to maintain multiple levels of security within a system. By assigning a security label to a resource, the security administrator can prevent the movement of data from one level of security to another, that is, declassification of data.

3.5.2 Mandatory Access Control

MAC is the principle of restricting access to objects based on the sensitivity of the information that the object contains and the authorization of the subject to access information within that level of sensitivity. It is mandatory since subjects cannot control or bypass it. The sensitivity of objects is defined by the security administrator, not the owner of the data. The security administrator will need to be someone with the RACF system SPECIAL attribute. The security label indicates the hierarchical level of classification of the information (such as top secret, sensitive, or unclassified), and also inherently indicates to which non-hierarchical category the information belongs within that level (such as project A, project B, or project C). The security administrator also controls each subject's access to an object or data within the multi-level secure system by specifying which security labels the subject can use. A subject can access information in an object only when the subject's security label entitles the access. If the subject's security label does not have enough authority, the subject cannot access the information in the object.

Mandatory Access Control is based on the theory of *dominance* between security labels. This dominance is based on each security label's combination of security level and zero or more categories.

3.5.3 Discretionary Access Control

DAC is the principle of restricting access to objects based on the identity of the subject (the user or the group to which the user belongs). Discretionary Access Control is implemented using *access control lists*. A RACF resource profile contains an access control list that identifies the users who can access the resource and the authority (such as read or update) the user is allowed in referencing the resource. The access control list overrides the universal access list of the resource profile. The security administrator defines a profile for each object (a resource or group of resources), and updates the access control list for the profile. This type of control is *discretionary* in the sense that subjects can manipulate it, because the owner of a resource, in addition to the security administrator, can identify who can access the resource and with what authority.

3.5.4 Security levels and security categories

Security labels establish an association between a RACF *security level* and a set of zero or more RACF *security categories*. For example, a system might have three security levels, such as top secret, sensitive, and unclassified. It could have various security categories that span these security levels and could represent individual projects or departments, for example, project A, project B, and project C.

Security level This is controlled via a resource profile called SECLEVEL in the RACF class SECDATA. It defines the hierarchical degree of sensitivity of the data. The security administrator defines in RACF each level that comprises a name and a numerical security level (the higher the number, the higher the security level). In the above example you might define top secret as level 30, sensitive as level 20, and unclassified as level 10. The security administrator can define up to 254 different levels, although this would be an impractical amount. We recommend keeping the setup simple and concise. For details on how to define the security level see 3.5.5, “Defining security labels” on page 75.

Security category This is controlled via a resource profile called CATEGORY in the RACF class SECDATA. It is non-hierarchical and further qualifies the access capability. The security administrator can define zero or more categories that correspond to some grouping within an organization that has similar security classifications. For details of how to define security categories see 3.5.5, “Defining security labels” on page 75.

Security label After defining the SECLEVEL and CATEGORY profiles above, the security administrator defines a resource profile in the class SECLABEL for each security label. The security label is a name of up to eight uppercase alphanumeric or national characters. The national characters are $(X’5B’), #(X’7B’), and @(X’7C’). The first character cannot be numeric. Each security label name must be unique. Each SECLABEL profile defines a combination of a SECLEVEL member and zero or more members of the CATEGORY profile, which are required for that particular security label. Note that you do *not* need to define a security label for *every* possible combination of level and category. For details on how to define security categories see 3.5.5, “Defining security labels” on page 75.

Guideline: Although the system allows the definition of several thousand categories, up to 254 security levels, and unlimited security labels, define *only* as many as you really need. A large number of levels and categories can decrease performance, particularly at IPL time and for the SETROPTS RACLIST(REFRESH) command. DB2 caches security labels (per commit scope) to avoid extra calls to RACF. While the impact is not measured yet, it is expected to be small. The caching would work best if there are a relatively small number of security labels to be checked compared with the number of rows accessed in a long commit scope.

Security labels that the system creates

The system creates four labels automatically at initialization time.

- SYSHIGH: This label is equivalent to the highest security level defined by the security administrator, and all categories defined by the security administrator. It dominates all other security labels in the system. It will always be of a higher level than any other in the system. SYSHIGH should be restricted to special system-level address spaces such as consoles, and to system programmers, system operators, and system administrators.
- SYSLOW: This label is equivalent to the lowest security level defined by the security administrator, and has no categories. It is dominated by all other security labels. SYSLOW should only be used for resources that have no classified data content. It is appropriate to use SYSLOW for data sets that IBM supplies for which the following is true:
  - Most users only need to read them.
  - A limited number of users, such as system programmers, might need to update them. They should only do so when running at a very low classification, to prevent them from accidentally putting classified data into the data sets.

  If a resource is not in a class that requires reverse mandatory access checks or equal mandatory access checks, assigning a security label of SYSLOW to the resource allows all subjects to pass a mandatory access check for read access to the resource. Subjects still must pass the discretionary access check in order to access the resource.
- SYSNONE: This label is treated as equivalent to any security label to which it is compared. SYSNONE, like SYSLOW, should only be used for resources that have no classified data content. It is different from SYSLOW in that all users might need to update resources with SYSNONE, even when running at a high classification. It is intended for use on resources that must be written to different security labels when write-down is not allowed. It is used to ensure that a user is permitted read/write access to a data set such as a catalog.

  Follow these guidelines for assigning the SYSNONE security label:
  - Use SYSNONE for a data set only when some other process (such as catalog management, or a program via program access to data sets (PADS)) mediates the user's access to ensure that no classified data is written into the data set.
  - Use SYSNONE for a z/OS UNIX file only when you limit discretionary access to the file to a specific UID, and the only access to the file is via a z/OS UNIX program with the setuid option that switches to that UID and ensures that no classified data is written into the data set.
  - Do not use SYSNONE for users.

- SYSMULTI: This label is considered to be equivalent to *any* defined security label. It is intended for use by the following:
  - Server or daemon address spaces whose implementation and documentation explicitly support Multilevel Security, giving them the ability to perform and separate work for users running with different security labels
  - zFS directories that can contain data with different security classifications, such as the root directory of a file system

  Follow these guidelines for assigning the SYSMULTI security label:
  - SYSMULTI is not an appropriate security label for a data set or z/OS UNIX file, unless access to the data set is mediated via PADS or some other mechanism to ensure that either of the following is true:
    - Users can only write, and not read.
    - Users can only read appropriate parts of the data.
  - SYSMULTI is not generally appropriate for users.

3.5.5 Defining security labels

Once you activate the SECLABEL class, users who log on without a security label can no longer access resources protected by security labels. Therefore, you should assign a default security label to all users before you assign a security label to a commonly accessed resource, such as SYS1.BRODCAST. Each user who does not have a default security label must specify a security label at logon time or else be denied access to the system. If you assign a security label to a resource profile while the SECLABEL class is active, the security label you assign to the users must allow the required users to access the resource.

Before defining the SECLABELs, the security administrator must also define two profiles in the RACF SECDATA resource class: One to define the security levels, and the other to define the security categories for the system.

The security labels will be defined in RACF and used in a special column of the DB2 tables to define the security classification of the rows in those tables. The RACF definitions will be in the SECLABEL resource class. This class will be known to RACF by virtue of being defined in the RACF Class Descriptor Table, and you should activate it after you have defined your system's security labels. The class should also be RACLISTed for best performance, which will mean that the profiles are held in memory.

The resource class SECLABEL should be active and RACLISTed. A subject will need to have at least one security label defined as its default; they can have authorization to more than one security label, but can only use one security label at a given time. Authorization to one or more security labels is achieved by being permitted READ access to the resource in the SECLABEL class.

3.5.6 Authorizing users to access security labels

Authorization to use a security label is given by permitting the user READ or higher access to the security label profile in the resource class SECLABEL. Users can have authority to use more than one security label, but remember, as we have already said, a user can only use one security label at any one time, as defined at logon time. To change the security label the user will need to log off and log on with another security label. A user will usually, but not necessarily, have a default security label defined to his user ID. Assigning a security label to a new user can be done when the user ID is initially defined with the ADDUSER command, or for an existing user it is done by altering the user ID with the RACF ALTUSER command.

A security label cannot be assigned to a RACF group; only individual subjects or objects can have security labels assigned to them. However, it is important to understand that authorization to use a security label can be given by permitting a RACF group (to which the user must be connected) READ access to the security label profile in the resource class SECLABEL, that is, via the access list of that security label. See also the tip below.

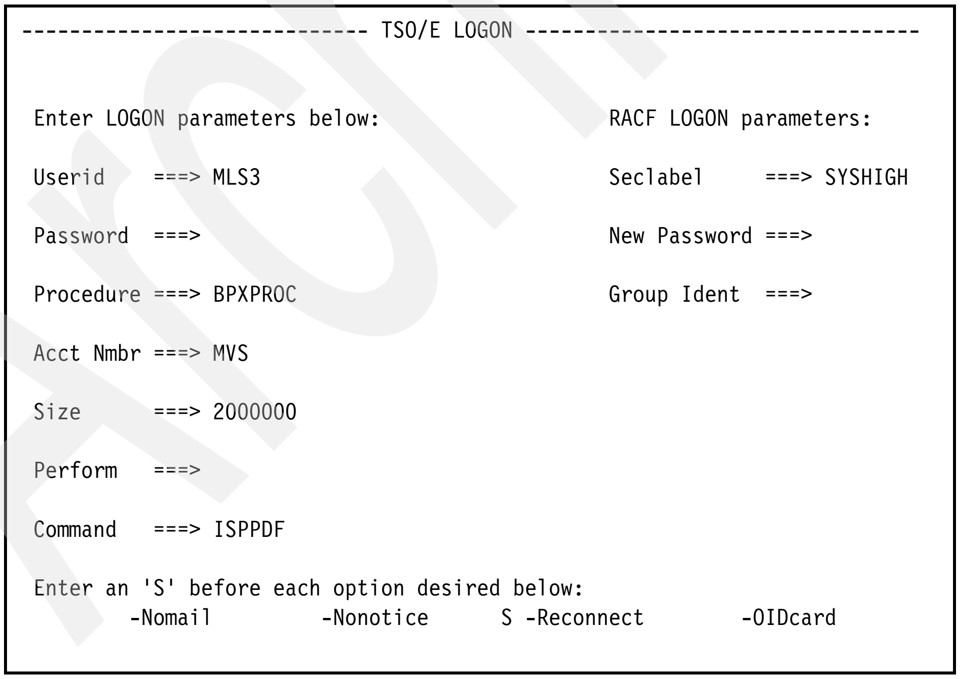
Access to a security label is given by permitting the user to the security label profile in the resource class SECLABEL. A user will usually not necessarily have a default security label defined to his user ID. Do this using the RACF alter user command:

```
ALTUSER USER1 SECLABEL(XYZ)
```

When the SECLABEL class is activated, the TSO/E LOGON panel will have an extra input field for specifying the security label. This will show the user's default security label. See Figure 3-3.

A user will be permitted access to zero or more security labels. On the logon panel she can specify the security label to which she has read access. She will be governed by that security label during that session.

Tip: A useful way to administer access to SECLABELs with large numbers of users is to create a RACF group of the *same name* as the SECLABEL; permit that group read access to the SECLABEL and then give them read access via connecting users to the group. For numbers of users greater than 1024 you should consider defining the group as a universal RACF group. This use of groups keeps access lists to a manageable size and also provides an immediate indication, when listing the user, of which SECLABELs a user has access to without needing to run a RACF report utility IRRUT100.



```
----------------------------- TSO/E LOGON ---------------------------------


 Enter LOGON parameters below:                       RACF LOGON parameters:

 Userid    ===> MLS3                                 Seclabel     ===> SYSHIGH

 Password  ===>                                      New Password ===>

 Procedure ===> BPXPROC                              Group Ident  ===>

 Acct Nmbr ===> MVS

 Size      ===> 2000000

 Perform   ===>

 Command   ===> ISPPDF

 Enter an 'S' before each option desired below:
         -Nomail         -Nonotice      S -Reconnect        -OIDcard
```

Figure 3-3 TSO logon screen

3.5.7 Using security labels

After security labels have been created and assigned, the security administrator can activate the RACF SECLABEL resource class to cause the system to use the security labels for authorization checks. Then, when a user tries to access a resource, RACF checks whether the resource has a security label. If it does, RACF compares the security label of the user with that of the resource (this is Mandatory Access Control (MAC)). If the security labels allow access, RACF then checks the access list of the profile that protects the resource (this is Discretionary Access Control (DAC)). The decision as to whether to allow the access is based on MAC, based on security labels; and DAC, based on access lists.

To activate the SECLABEL class, the security administrator issues the command:

```
SETROPTS CLASSACT(SECLABEL) REFRESH RACLIST(SECLABEL)
```

Note that when the SECLABEL class is active, unless the security administrator has also set certain system options, a user needs a security label only if the resource the user wants to access has a security label, and resources are not required to have security labels. To increase security, the security administrator can use the SETROPTS command to set RACF system options that require that certain resources have security labels, and require that any user who tries to access those resources has a security label.

3.5.8 Dominance

In this section we review dominance.

Comparing security labels

When authorization checks are made to determine security label authorization, the relationship between security labels is assessed. The types of these relationships are:

- Dominance
- Equivalence
- Disjoint

One security label dominates a second security label when the following two conditions are true:

- The security level, defined in class SECDATA, that is associated with the first security label is *greater than or equal to* that of the second security label.
- The set of security categories, defined in class SECDATA, that are associated with the first security label *includes* the set of security categories associated with the second security label.

If neither security label dominates the other then the two security labels are said to be *disjoint* or *incompatible.*

Table 3-1 gives an example of security label dominance.

Table 3-1 Security levels example

Security level	Hierarchical level
Top secret	L3
Sensitive	L2
Unclassified	L1

Table 3-2 Security categories example

Security categories
A, B, C, D, and E

Table 3-3 Security labels and their levels and categories

SECLABEL	SECLEVEL	CATEGORY
L3ABCDE	Top secret	A, B, C, D, and E
L2BC	Sensitive	B and C
L1AB	Unclassified	A and B
L1C	Unclassified	C

Here L3ABCDE dominates L2BC because the hierarchical security level of TOP SECRET (L3) is higher than that of SENSITIVE (L2), and L3ABCDE's non-hierarchical categories include all of L2BC's categories.

Table 3-4 Security labels showing seclevels and categories

	A	B	C	D	E
Top secret	seclabel=L3ABCDE				
Sensitive	*no label*	seclabel=L2BC		*No label*	*No label*
Unclassified	seclabel=L1AB		seclabel= L1C	*No label*	*No label*

L3ABCDE also dominates L1AB and L1C.

L2BC dominates L1C.

L2BC and L1AB are disjoint.

See also Figure 3-5 on page 79.

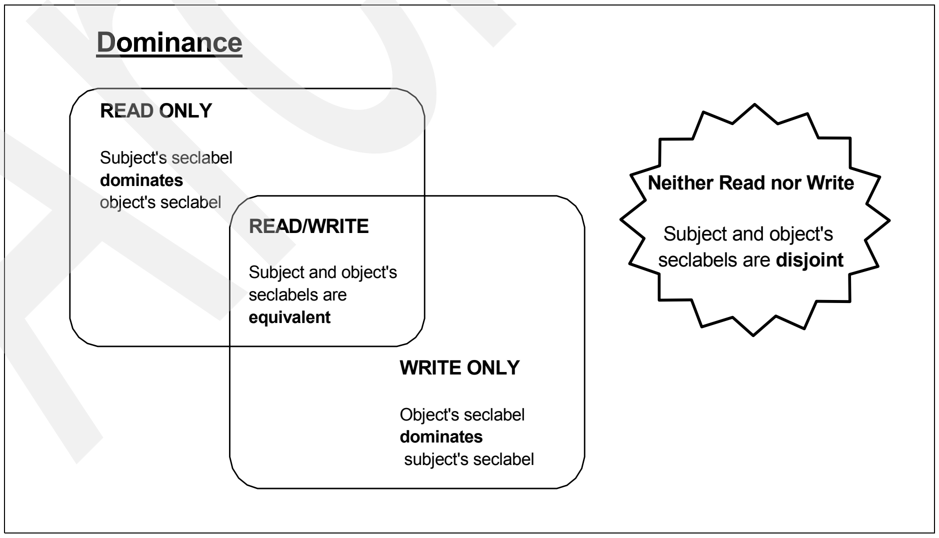

Dominance
READ ONLY
Subject's seclabel **dominates** object's seclabel
READ/WRITE
Subject and object's seclabels are **equivalent**
WRITE ONLY
Object's seclabel **dominates** subject's seclabel
Neither Read nor Write
Subject and object's seclabels are **disjoint**

Figure 3-4 Seclabel dominance schematic1 of 2

This shows, in their simplest terms, the relationships between the four cases of subject being either:

- Dominating the object - READ ONLY
- Equivalent to the object - READ/WRITE
- Dominated by the object - WRITE ONLY
- Disjoint - None of the above

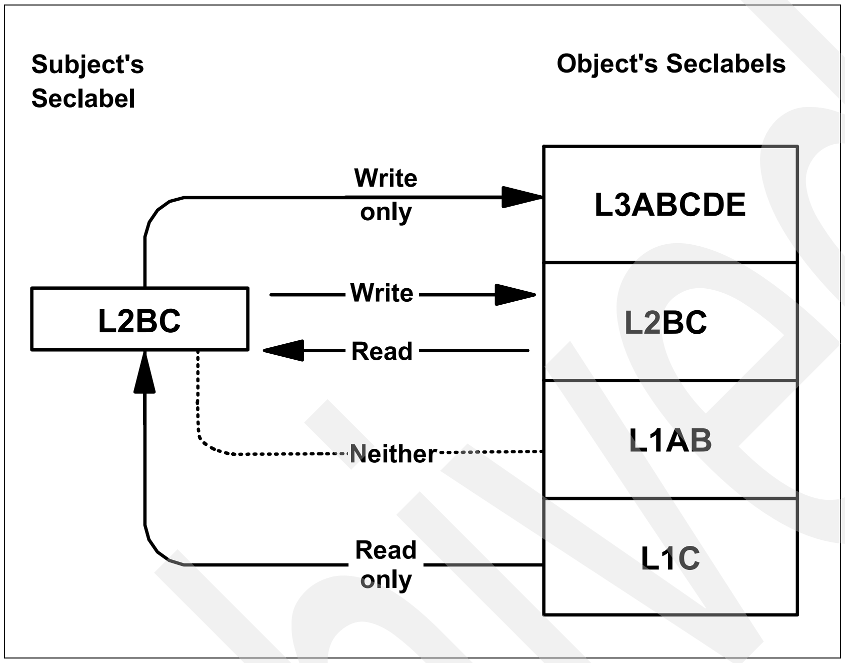


Figure 3-5 Seclabel dominance schematic 2 of 2

This graphically shows our example. In this example it is important to note that the user does not have the ability to write down. This is to say that MLS(FAILURES/WARNING) is not set or MLS(FAILURES/WARNING) is set, but they do not have access to the RACF profile IRR.WRITEDOWN.BYUSER in the FACILITY class. The user has a security clearance of SENSITIVE, but only for categories B and C, and so will have access to a security label of L2BC.

3.5.9 Security label authorization checking

When the SECLABEL class is active on your system, and a subject requests access to a resource, RACF compares the security label of the subject with that of the resource. For a general description of these comparisons, see “Comparing security labels” on page 77.

1. If the subject requesting access does not have a security label and the resource does have a security label, RACF fails the request.
2. If the SETROPTS MLACTIVE(FAILURES) option is in effect and the resource does not have a security label associated with it, and the resource class is DATASET or another class that requires security labels as defined in the RACF class descriptor table (CDT), RACF fails the request.
3. If the SETROPTS MLACTIVE(WARNING) option is in effect, RACF makes the same checks as in step 2. If the access check fails because the resource does not have a security label, RACF issues a warning message and grants the request.
4. If the SETROPTS MLS(FAILURES) option is in effect, RACF makes the tests in the following table, and fails the request if the test fails.

Requested access level	Normal MAC	Reverse MAC	Equal MAC
Read-only	Subject dominant	Object dominant	Equivalent
Read/write	Equivalent	Equivalent	Equivalent
Write-only(1)	Object dominant(2)	Unpredictable(3)	Equivalent

Security label authorization checking when SECLABEL class is active and either SETROPTS MLS(FAILURES) or MLS(WARNING) is in effect.

Notes:
(1) z/OS does not support write-only requests for data sets or tape volumes. All write-only requests are tested as both read-only and write-only requests. Therefore, the security labels must be equivalent.
(2) Subjects cannot write to an object that has a lower security label than the subject's security label. This inability to write down is enforced when SETROPTS MLS(FAILURES) is in effect to ensure that a subject does not declassify data.
(3) The test for write-only is not supported for classes defined with the reverse MAC attribute.
If the SETROPTS COMPATMODE option is in effect, RACF checks to see if the subject's UTOKEN indicates that the ACEE was created with an older protocol (pre RACF 1.9). If both are true, then RACF checks to see if the subject has access to a security label that could allow the requested access to the object.

If the subject has access to any such security label, RACF fails the request.

If the subject does have access to such a security label, RACF continues authorization checking and logs the request.

5. If the SETROPTS MLS(WARNING) option is in effect for this resource class, RACF makes the same checks as in step 4. If any test fails the request, RACF issues a warning message and grants the request.
6. If the SETROPTS NOMLS option is in effect, RACF makes the tests in the following table and fails the request if the test fails.

Requested access level	Normal MAC	Reverse MAC	Equal MAC
Read-only	Subject dominant(2)	Object dominant	Equivalent
Read/write	Subject dominant(2)	Object dominant	Equivalent
Write-only(1)	Object dominant	Unpredictable(3)	Equivalent

Security label authorization checking when SECLABEL class is active and either SETROPTS NOMLS is in effect or the subject is in write-down mode.

Notes:
(1) z/OS does not support write-only requests for data sets or tape volumes. All write-only requests are tested as both read-only and write-only requests. Therefore, the security labels must be equivalent.
(2) If SETROPTS MLS(WARNING) is in effect instead of NOMLS in these cases, an ICH408I warning message is written.
(3) The test for write-only is not supported for classes defined with the reverse MAC attribute.
If the SETROPTS COMPATMODE option is in effect, RACF checks to see if the subject's

UTOKEN indicates that the ACEE was created with an older protocol (pre RACF 1.9). If both are true, then RACF checks to see if the subject has access to a security label that could allow the requested access to the object.
If the subject has access to any such security label, RACF fails the request.

If the subject does have access to such a security label, RACF continues authorization checking and logs the request.

If the object is a JES spool data set, RACF uses the security label in the token associated with the data set (specified on the RTOKEN) parameter of the RACROUTE REQUEST=AUTH macro). Otherwise, RACF uses the security label kept in the resource profile protecting the object, in the FSP for files, or in the ISP for IPC objects.

3.5.10 Using system-specific security labels in a sysplex

In a sysplex it can be useful to limit the use of certain security labels to certain members of the sysplex. This allows one member of the sysplex to run work at security label A, while another handles work at security label B, keeping work separated based on security classification while still sharing the RACF database. The SETROPTS option SECLBYSYSTEM allows you to use security labels on a per-system basis.

To define system-specific security labels, the security administrator specifies on which systems a security label is to be active by adding a member list to the SECLABEL resource class profile. The member names are system SMF IDs containing 1–4 characters. For example, to define the security label named SECRET as being active only on the systems with SMF system IDs SYSA and SYSB, the security administrator could define SECRET with a command like:

```
RDEFINE SECLABEL SECRET....ADDMEM(SYSA,SYSB)
```

If no member list of system IDs is added, the security label is considered to be active on all systems sharing the RACF database.

To activate the use of system-specific security labels, activate the SECLBYSYSTEM option and refresh the SECLABEL class:

```
SETROPTS SECLBYSYSTEM
SETROPTS RACLIST(SECLABEL) REFRESH
```

The following restrictions apply to system-specific labels:

- JES3 does not support the use of system-specific security labels. Do not activate the SECLBYSYSTEM SETROPTS option if you are using JES3.
- JES2 does not support the use of system-specific security labels for systems that perform NJE and OFFLOAD processing. These systems must have all security labels active. In addition, JES2 printers cannot process output unless the security label associated with the output is active on the system controlling the printer.
- If you define system-specific security labels, be aware that using a generic TSO system name at logon might not work, because the user could not be allocated to a system where the user’s security label is not active.
- If you use Application Restart Manager (ARM) to manage applications and you use system-specific security labels, ensure that the systems that you have told ARM to use when restarting an application are systems that have the appropriate security label active. Otherwise, ARM might try to restart an application requiring a particular security label on a system where that security label is not active, and the application restart will fail.

MLS applied to TCP/IP communications

This chapter provides a simple example of how to set up and use MLS to control access to TCP/IP resources and the data flowing through these resources.

Books to be used for references are:

- *z/OS Communications Server IP Configuration Guide*, SC31-8775
- *z/OS Communications Server IP Configuration Reference*, SC31-8776

4

MLS as applied to TCP/IP communications

This chapter provides a simple example of how to set up and use MLS to control access to TCP/IP resources and the data flowing through these resources.

Books to be used for references are:

- *z/OS Communications Server IP Configuration Guide*, SC31-8775
- *z/OS Communications Server IP Configuration Reference*, SC31-8776

The environment definitions to achieve MLS networking in z/OS are mainly based on the use of RACF profiles in the SERVAUTH class. These profiles were already available before extending the MLS support to TCP/IP resources at z/OS 1.5. z/OS 1.5 brings the capability of associating a security label and a port of entry to these resources.

4.1 z/OS TCP/IP and the SERVAUTH class

The protection of TCP/IP resources such as stacks, ports, and networks can be achieved using profiles in the RACF SERVAUTH class. TCP/IP is then acting as the resource manager, which uses the SAF interface to have the requestor's permission to the resource evaluated by RACF.

The profiles in the SERVAUTH class we are interested in with MLS are the profiles protecting access from, and optionally to, a network and to a local z/OS TCP/IP stack. They have not been implemented for MLS only, and we will look first, as an introduction, at their initially intended use. Then we will see how they can be used to contribute to the establishment of a multi-level security environment for z/OS TCP/IP communications.

Use of SERVAUTH profiles in a non-MLS environment

The SERVAUTH class of profiles was introduced in OS/390 2.6, as a means of protecting resources managed by the OS/390 TCP/IP stack. The range of resources protected in this class has been extended along the further releases of OS/390 and z/OS, and includes, as of z/OS 1.6 today, the following resources:

- The TCP/IP stacks running in the z/OS image
- The networks the z/OS image has connectivity to
- The ports reserved for permitted applications
- The Netstat command and its options
- The policy agent pasearch command
- The FTP SITE DUMP AND DEBUG commands
- Usage of SNMP subagents that connect to the TCP/IP SNMP agent
- The MODDVIPA utility
- The Fast Response Cache Accelerator (FRCA)
- TCP connection information service intended for network management applications and the selection of SMF records
- The TCP/IP packet trace service access control
- The HFS as accessed by FTP

Again, we focus here on the protection of TCP/IP stacks and networks.

4.1.1 Stack access control

Access to a given TCP/IP stack (remember that z/OS supports up to eight concurrent stacks) is controlled with a profile in the SERVAUTH class. The profile's name has the following form:

```
EZB.STACKACCESS.sysname.tcpproc
```

Where EZB.STACKACCESS is a fixed value, *sysname* is the value of the system static symbol &SYSNAME, and tcpproc is the name of the protected stack started procedure. An example is given in Figure 4-1 on page 85. In this example, two stacks are being protected with STACKACCESS profiles—the stack started with an STC called TCPIPA and the stack started with STC TCPIPB. Both profiles are defined with a UACC none, and users with the RACF user ID USER1 and USER2 are eventually given READ access to these resources. In this example the application running with the user ID USER2 will be denied access to the stack TCPIPA, as only USER1 is granted access to this stack.

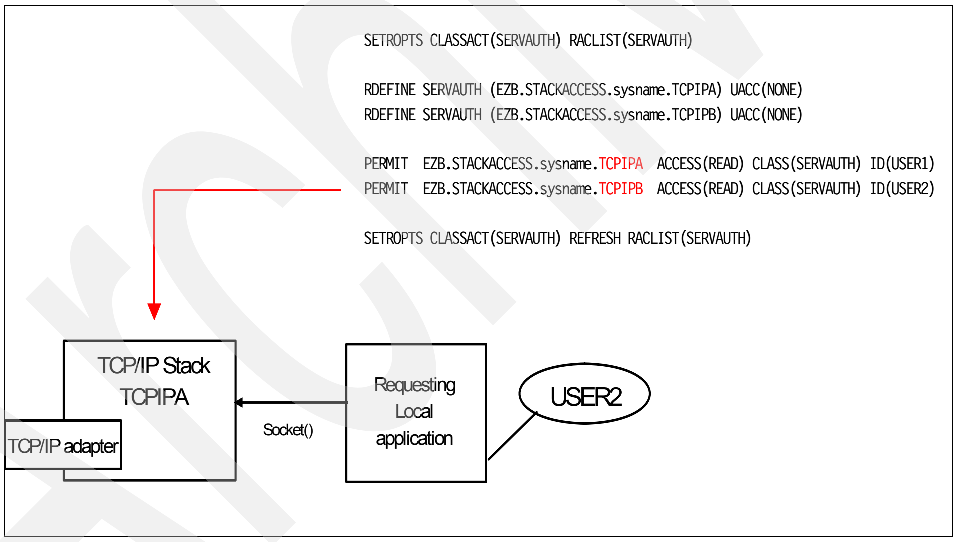
TCP/IP acts as a resource manager and requests SAF, provided that the SERVAUTH class is active, to check for permission of an application to issue the socket() call to the stack. The user ID considered for access control checking is:

- For a TSO user: The tso user ID.
- For an OMVS shell user: The effective user ID of the process (this is normally the same user ID as the logged-on user, but can also have been changed via the setuid() or seteuid() functions.
- For a batch job: The user ID associated with the batch job via USER= in the job JCL or by RACF.
- For processes started from the MVS console: The user ID associated with the procedure by the STARTED class of profiles in the RACF database.

Failing the authorization is shown by a RACF message on the system console. The application's behavior remains application dependant.

Note: TCP/IP internal tasks (except TN3270 with NACUSERID configured), VTAM, and the USS kernel are exempt, by design, from STACKACCESS checking.

Note: All programs that need to access the stack (including utilities such as Netstat, Ping, etc.) must have a user ID permitted to the STACKACCESS profile, if one has been defined.


```
SETROPTS CLASSACT(SERVAUTH) RACLIST(SERVAUTH)

RDEFINE SERVAUTH (EZB.STACKACCESS.sysname.TCPIPA) UACC(NONE)
RDEFINE SERVAUTH (EZB.STACKACCESS.sysname.TCPIPB) UACC(NONE)

PERMIT  EZB.STACKACCESS.sysname.TCPIPA  ACCESS(READ) CLASS(SERVAUTH) ID(USER1)
PERMIT  EZB.STACKACCESS.sysname.TCPIPB  ACCESS(READ) CLASS(SERVAUTH) ID(USER2)

SETROPTS CLASSACT(SERVAUTH) REFRESH RACLIST(SERVAUTH)
```

Figure 4-1 Example of STACKACCESS setup

MLS support

What has been described above is the DAC piece of stack access control. For MAC to be enforced by the TCP/IP stack, the RACF STACKACCESS profile can be given a security label. This can be achieved by the RACF command below:

```
RDEFINE SERVAUTH (EZB.STACKACCESS.sysname.tcpproc) + SECLABEL(seclabel_name)
UACC(READ)
```

The user ID the stack is running under must have this SECLABEL defined as its default SECLABEL, and must be permitted in READ to this SECLABEL profile.

Note that the universal access granted here in the STACKACCESS profile is READ, meaning that only MAC is to be enforced. It could be any other value if DAC enforcement is also needed to be subsequently in effect.

4.1.2 Network access control

Network access control implements the concept of *security zones*. A network security zone is an administrative name for a collection of systems that require the same access control policy. IP addresses are used to map systems into security zones.

The capability of sending packets to, or receiving packets from, a network security zone is controlled with NETACCESS profiles in the RACF SERVAUTH class. The TCP/IP stack checks that local applications are authorized to exchange information with these networks on the basis of their user ID's permission to these profiles.

Definition of the network security zones

A NETACCESS and an ENDNETACCESS statement are used to specify which network accesses have to be protected by RACF in the TCPIP.PROFILE data set. Here is an example of such a specification:

```
NETACCESS INBOUND OUTBOUND
192.168.0.0/ 16 SUBNET1 ; Subnet address
192.168.113.19/ 32 HOST1 ; Specific host address
192.168. 113.0 255.255.255.0 SUBNET2 ; Subnet address
192.168. 112.0 255.255.248.0 SUBNET3 ; Subnet address
DEFAULT 0 DEFZONE ; Optional Default zone
ENDNETACCESS
```

In these definitions, network security zones are specified as IP addresses and subnet masks and are given a SAF resource_name: SUBNET1, HOST1, SUBNET2, etc. The DEFAULT keyword is used to specify any other IP addresses not explicitly specified in these statements, and in this example is linked to the SAF resource_name DEFZONE. The SAF resource_name is mapped to profiles in the SERVAUTH class with profile names of the form:

```
EZB. NETACCESS. sysname. tcpname. saf_resname
```

Attention: A word of caution here on possible confusion between the NETACCESS *statements* in the PROFILE.TCPIP and the EZB.NETACCESS *profiles* in the RACF database.

TCP/IP checks that the application that is to access the network to send or to receive data has READ permission to the corresponding EZB.NETACCESS RACF profile.

Note: These statements can be dynamically established at a TCP/IP stack using an OBEYFILE. The TCPIP.PROFILE data set can then be edited for permanent statements.

Note: The IP address mapped to a security zone can be a physical adapter address or a VIPA.

The INBOUND or OUTBOUND keywords can be used in the NETACCESS statement to indicate whether the network IP address being controlled is to be considered as a destination or origin IP address, the default being NOINBOUND OUTBOUND.

Access to networks specified with OUTBOUND is controlled when the application calls the following services:

- TCP connect, write
- UDP connect, write
- RAW connect, write

Access to networks specified with INBOUND is controlled when the application calls for:

- TCP accept, read
- UDP read
- RAW read

Some notes on the OUTBOUND and INBOUND condition follow.

OUTBOUND

Some applications, like FTP, start a new instance running under the authenticated user ID of the requesting user. Network access control is then exercised using this user ID when the new FTP instance communicates with the client.

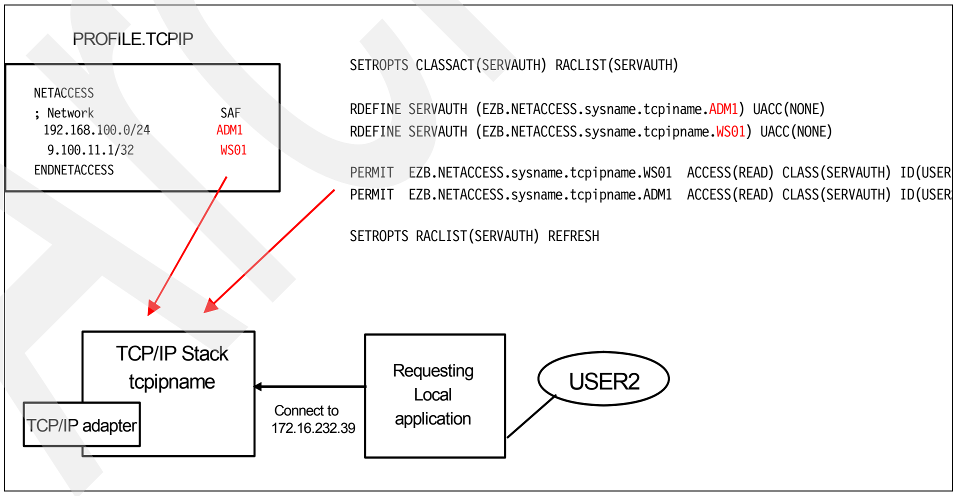
INBOUND

This is intended for applications that are to transfer the received data to another network, not actually sending data back to the original requestor. INBOUND sets control on which networks such an application can receive data from. It also controls an application's ability to bind() to a local IP address.

Pending TCP connections from unauthorized source addresses are silently reset and discarded during the accept processing. RAW and UDP datagrams from an unauthorized source address are silently discarded during the read processing.

The application, if not permitted to the network security zone, receives the following error code, which may show up in its log or trace:

```
ERRNO = 111 ñ 29452 (JRNetAccessDenied - Userid is not authorized to access the network)
```



```
NETACCESS
; Network                 SAF
  192.168.100.0/24        ADM1
   9.100.11.1/32          WS01
ENDNETACCESS
```

```
SETROPTS CLASSACT(SERVAUTH) RACLIST(SERVAUTH)

RDEFINE SERVAUTH (EZB.NETACCESS.sysname.tcpiname.ADM1) UACC(NONE)
RDEFINE SERVAUTH (EZB.NETACCESS.sysname.tcpipname.WS01) UACC(NONE)

PERMIT  EZB.NETACCESS.sysname.tcpipname.WS01  ACCESS(READ) CLASS(SERVAUTH) ID(USER
PERMIT  EZB.NETACCESS.sysname.tcpipname.ADM1  ACCESS(READ) CLASS(SERVAUTH) ID(USER

SETROPTS RACLIST(SERVAUTH) REFRESH
```

Figure 4-2 Example of NETACCESS setup

Exempt applications

The Network Access Control user ID is based on the application's address space information. TCP/IP is an exempt user and has access to all zones. Because Telnet runs in the TCP/IP address space, Telnet processes are also exempt from Network Access Control based on address space user ID information, unless the TN3270 ports have been defined with NACUSERID.

MLS support

What has been described above is the DAC piece of network access. For MAC to be enforced by the TCP/IP stack, the RACF NETACCESS profile can be given a security label. This can be achieved by the RACF command below:

```
RDEFINE SERVAUTH (EZB.NETACCESS.sysname.tcpproc) + SECLABEL(seclabel_name) UACC(READ)
```

Important: If no INBOUND and OUTBOUND keywords are stated, the default is NOINBOUND OUTBOUND. INBOUND OUTBOUND *must* be specified on MLS systems.

4.1.3 The notion of port of entry (POE)

As the name implies, the port of entry (POE) is an indication on how the request has entered the system, which RACF can exploit in its decision making by considering the POE as part of its permission criteria. A typical example of the latter is the TERMINAL class that can be used to identify a POE eventually considered for conditional access, as shown in the following example:

```
PERMIT SUBMIT.*.PAYROLL*.* CLASS(JESJOBS) ID(USER01) ACCESS(READ)
WHEN(TERMINAL(terminal-ID))
```

In this example USER01 is permitted to submit the job only if he entered the system via the terminal designated by terminal_ID.

As of z/OS 1.5, the POE can be the partner security zone, represented by the name of a corresponding NETACCESS profile in the SERVAUTH class. This can be taken into account for conditional access, such as:

```
PERMIT ‘MLS4.TEST.L3C’ CLASS(DATASET) ID(USER01) ACCESS(READ) +
WHEN(SERVAUTH(netaccess_profile))
```

The port of entry can also be used to associate a SECLABEL to the process or thread instantiated to serve a user request, such as when a user logs in to the z/OS FTP server. When FTP.DATA contains the statement `PORTOFENTRY4 SERVAUTH` and the client’s IP address is mapped to a NETACCESS security zone, then the NETACCESS profile is remembered as the POE. If MAC is enforced and the FTP user is permitted to the SECLABEL specified for the client’s security zone, then this very SECLABEL is associated to the request. This is demonstrated in the example given in “Our test” on page 96.

Note: The POE support is provided to applications or resource managers where the work entered the system with the following functions:

- __poe() or calls BPX1POE or BPX4POE prior to changing to client's login identity.
- SIOCGSOCKPOEATTRS ioctl and then use that information about RACROUTE VERIFY.

As of today, this is performed in FTPD, INETD (it covers all the daemons forked by INETD), and DB2 V8.

4.2 The MLS networking environment

In this section we discuss the MLS networking environment.

4.2.1 Some MLS basics (again)

As already seen in the previous chapters, MAC is based on the domination relationship between the accessor's and the resource's respective security labels.

Important notions to remember are:

- Domination: Security label A dominates security label B when it has an equal or higher security level and comprises all the categories of security label B.
- Equivalence: Security label A is equivalent to security label B when both security labels have the same security level and the same set of categories. Equivalent security labels dominate each other.
- Write down: Resources cannot, normally, be written down, that is, declassified. This is enforced by preventing a dominating accessor to write into the resource, whereas he can read from the resource.

Other useful definitions are:

- The SYSHIGH system-created security label: It always dominates any other defined security labels.
- SYSLOW: To the contrary, SYSLOW is dominated by any other defined security label.
- SYSNONE: A resource with the SYSNONE security label is not protected by MAC in RACF. It is expected that MAC is mediated by a the application providing access to the resource.
- SYSMULTI: This security label is equivalent to any other defined security labels.

In the networking environment, the information that is being protected is the data being read and written through sockets. Sockets are opened and used by applications running under user IDs. In a z/OS multilevel secure environment:

- Each user ID is permitted to use one or more security labels.
- Every job or login session is associated with a user ID.
- A user ID can use only one security label for each job or login session.
- The security label used must be limited by the port of entry (source type and location) of the job or login session. That is it is up to the application providing the service to insure that, if running with the SYSMULTI security label, it properly associates the POE security label to any access done on behalf of the client.

The conceptual flow of information, as seen from each side of a TCP/IP communication, is depicted in Figure 4-3 on page 90. In order for the data to flow both ways (that is, in read and write mode), all the security labels encountered along the data path should be equivalent. These are the security labels of:

- The stored data, as defined in the RACF data set profile or the file security packet. Note that security label equivalency is not a requirement here, as one may precisely want to exploit the no read-up and no write-down properties implied by MLS dominance.
- The application reading or writing the data, which is running under the security label associated to its user ID or the POE.
- The TCP/IP stack hosting the local socket. z/OS introduces the concept of restricted and unrestricted stack. This concept is explained later in this chapter.
- The packet's origin and destination addresses, with an explicit packet tag as explained in "Security label of the origin and destination addresses" on page 90; or implicitly assumed via network security zones.

- The partner application.

Note that the exchanged packets can flow though intermediate networks via routing devices, which are not affecting the origin and destination addresses in the packets. However, these intermediate devices may themselves generate packets intended for either end of the communication. These packets must also be associated to a security label in an MLS environment, by defining security zones to map the subnetworks these devices are in.

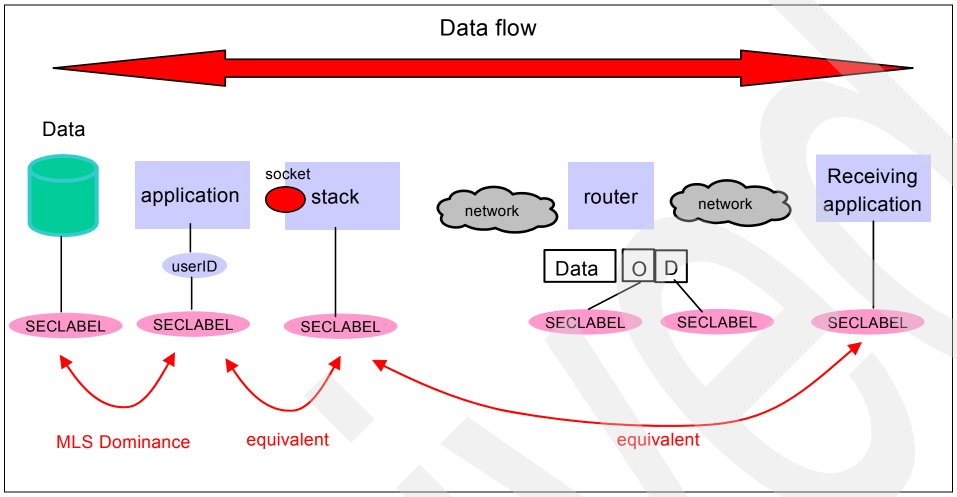


Figure 4-3 MLS network information flow

Security label of the origin and destination addresses

In this section we discuss the security label of the origin and destination addresses.

Packet tagging

In order to indicate the security labels of an IP packet's origin and destination addresses, IBM has developed a proprietary packet tag that is generated and exploited by the z/OS stacks in the following situations:

- The communicating stacks are located in the same z/OS image (environment variable IUTSAMEHOST).
- Or they are located in the same sysplex and communicate with IP over XCF.
- And the local and remote security zones have the SYSMULTI security label.

Note: During our trials we experienced what seemed at first to be a problem, but was actually working as designed: The receiving MLS systems had both their local and remote security zones with the SYSMULTI security label. It was definitely not answering to our client system, and not located in the same sysplex or same z/OS image. The MLS system was actually expecting to receive packet tags along with IP packets, because of these SYSMULTI definitions, and was silently discarding the packets because there were not any.

Security zones security labels

When the conditions required to generate packet tags are not met, then the security labels of the origin and destination addresses are the ones specified in the SERVAUTH NETACCESS profiles. Profiles are defined for the security zones corresponding to local IP addresses and security zones corresponding to the remote partners addresses. The stack then assumes these security labels to be the ones pertaining to origin and destination addresses. This is shown in Figure 4-4 on page 91.

The POE security label is inherited from the security label of the security zone the partner's IP address is mapped to, as specified in the RACF command:

```
RDEFINE SERVAUTH (EZB.NETACCESS.sysname.tcpiname.subnet_saf_name) + SECLABEL(seclabel_name)
UACC(READ)
```

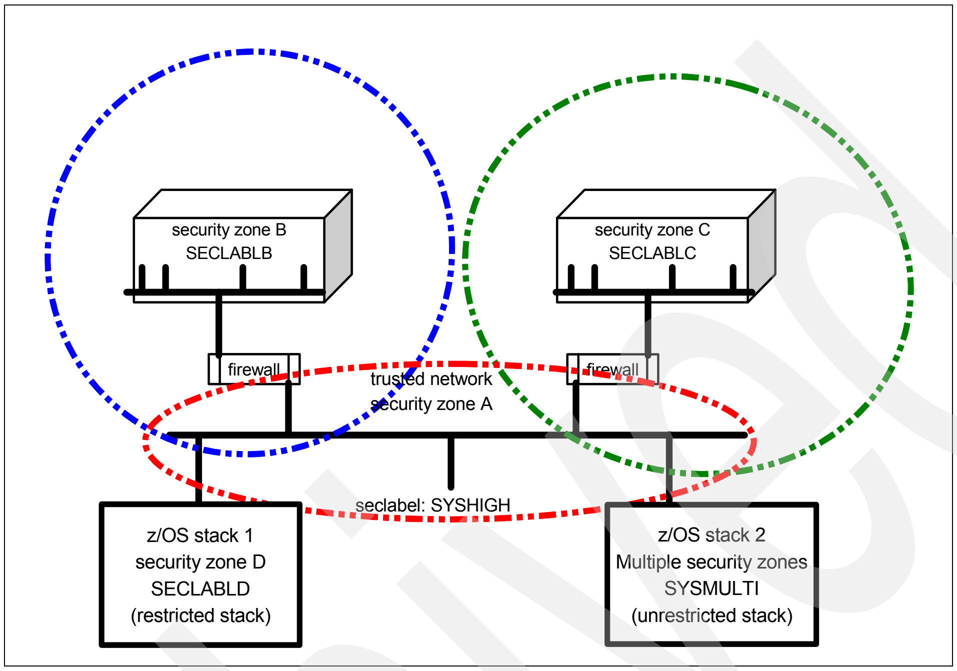


Figure 4-4 Assigning security zones with security labels

In Figure 4-4 the network, as seen by the z/OS TCP/IP stacks, has been defined as four security zones: A, B, C, and D (that is, sets of IP addresses covered by a common access policy). From the MLS standpoint, IP addresses in zone B are valid destinations for packets containing data with an equivalent security label to SECLABLB, or packets issued from these addresses have to contain data with an equivalent security label to SECLABLB. One z/OS stack, called a *restricted* stack, has its own IP addresses mapped to security zone D with SECLABLD, implying that data can only be exchanged with zones with an equivalent security label to SECLABLD. The other stack, the *unrestricted* stack, does not have any restriction with the zones it can communicate with, as its security label is SYSMULTI. The notions of restricted and unrestricted stacks are explained later in this chapter.

Security zone A is a special case in that it designates a trusted internal subnet, properly protected from the other networks by firewalls, with a security label of SYSHIGH, implying that packets issued in this subnet can only be exploited through a highly trusted data path with the SYSHIGH security label (or SYSMULTI, but this is not recommended from the security standpoint).

MLS enforcement and the SERVAUTH profiles

The MLS environment is enforced when the SECLABEL class and the MLACTIVE options are active in RACF. That is, applications and resources are given specific security labels and Mandatory Access Control is applied accordingly. The profiles in the SERVAUTH class:

- Must have a SECLABEL when SETR MLACTIVE.
- The class is defined in the CDT with EQUALMAC (that is, MAC access can be granted only for equivalent SECLABELs between the accessor and the resource).

A few more definitions

Now we review a few more definitions.

Nonsecure systems

Most systems do not support Mandatory Access Control processing as z/OS does. The security label of data flowing from and to these systems must be assumed. If these systems are not physically managed, they normally should not participate in a multilevel secure network. Some installations might need to permit them to participate. In these cases, it is recommended that they be assigned a security label, as seen from MAC supporting systems (that is, using the proper network security zone with z/OS), with the lowest security level and a single security category that is not common with any other security label.

Managed systems or single-level security (SLS) systems

Systems that do not support Mandatory Access Control processing can participate in a multilevel secure network, if they are physically managed to guarantee that all information about the system has the same single security label and all users of the system are permitted to that security label. These systems are referred to as single-level security (SLS) or managed systems in this chapter. This management requires both physical control of the systems and careful management of the network. Managed systems must be prevented from communicating with other managed systems that do not have equivalent security labels. Systems that support Mandatory Access Control and are configured to implicitly associate the correct security label with each managed system can also communicate with managed systems. The systems that perform Mandatory Access Control are responsible for ensuring that only information from applications with an equivalent security label is sent to a managed system, and that information received from a managed system is given only to applications with an equivalent security label.

As mentioned, physical control must be extended to the networking environment, so that:

- All interfaces on the subnet have the same security label.
- All packets generated within the subnet have a source address from the subnet.
- IP source routing options are not present.
- Communication is limited to other SLS subnets with equivalent security labels or to MLS systems.

Multilevel secure systems

Some systems in the network provide multilevel secure environments. These systems have mechanisms to associate security labels with information accessed through the system and with users logged into the system. This translates in z/OS into the use of packet tagging or network security zones.

The system enforces Mandatory Access Control policies to ensure proper separation of information. The packets being sent from a single IP address on the multilevel secure system might have originated from applications running under different security labels. Applications on a multilevel secure system can securely communicate with applications on a managed system.

Mandatory Access Control enforcement occurs only on the multilevel secure system. The multilevel secure system is responsible for ensuring that it sends only information from an application with an equivalent security label to any managed system. It also is responsible for ensuring that information received from a managed system is delivered only to an application with an equivalent security label. When two applications on multilevel secure systems communicate, the security label of the sending application must be communicated to the receiving system so that the receiving system can enforce Mandatory Access Control prior to delivering the information to an application.

Restricted stack

In the case of a restricted stack:

- The stack runs under a specific security label (as opposed to a system-defined security label) it inherits from its started task user ID.
- Accepts only user IDs with an equivalent label to use the stack, as the stack security label is also specified in a corresponding SERVAUTH STACKACCESS profile.
- Accepts packets with the IBM tag only if they have an equivalent security label.
- In the case of a missing IBM tag on the received packets (that is, packets coming from another TCP/IP host not located in the same z/OS image and not using IP over XCF), the packet inherits the security label of the NETACESS profile that maps the origin subnet of the packet, and must be of an equivalent security label to the stack security label.
- Packets are sent by this stack only to subnets with an equivalent security label to the security label in the NETACCESS profiles that map these subnets.

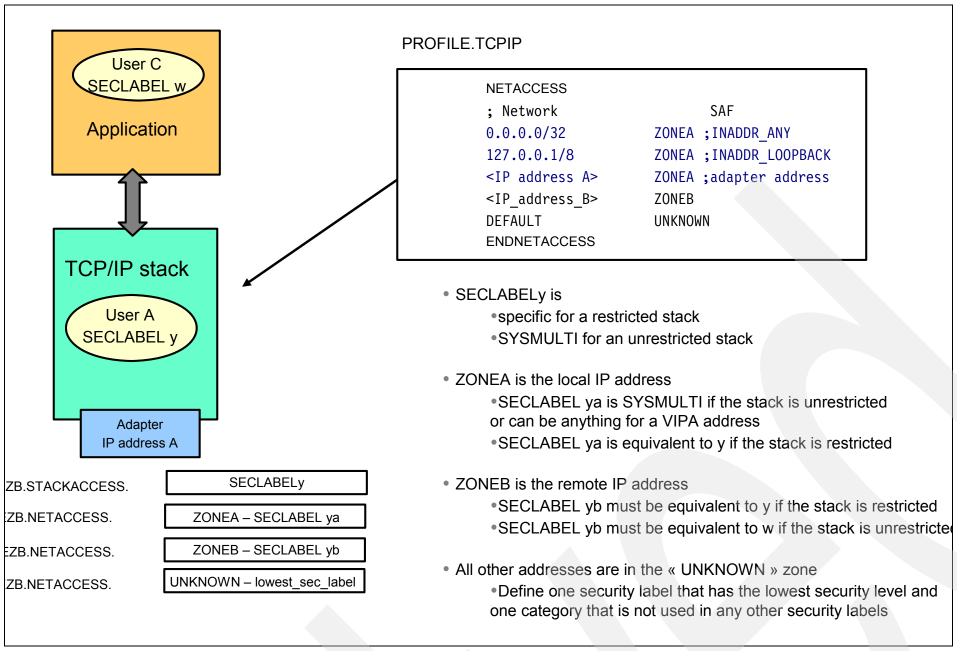
Because of all of the above, a restricted stack behaves like a SLS to the other TCP/IP hosts in the MLS environment.

Unrestricted stack

For an unrestricted stack:

- The stack runs under a user ID with the SYSMULTI security label.
- The stack allows sockets to be opened by applications with any security label.
- Unrestricted stacks are permitted to define VIPAs in network security zones with security labels other than SYSMULTI. When one of these is used as either the source or destination IP address of a packet, it implicitly identifies the security label of the information.
- When both IP addresses in a packet are in security zones with the SYSMULTI security label, the application security label must be explicitly transmitted in the packet, as it is the case with the IBM packet tag.

Figure 4-5 on page 94 summarizes the setup of a restricted and an unrestricted stack. In this figure the stack is running with the user ID A, which runs with security label y. The security zone A comprises the stack local IP addresses, whereas we are considering only one remote security zone: Zone B.



```
NETACCESS
; Network              SAF
0.0.0.0/32             ZONEA ;INADDR_ANY
127.0.0.1/8            ZONEA ;INADDR_LOOPBACK
<IP address A>         ZONEA ;adapter address
<IP_address_B>         ZONEB
DEFAULT                UNKNOWN
ENDNETACCESS
```

- SECLABELy is
  - specific for a restricted stack
  - SYSMULTI for an unrestricted stack
- ZONEA is the local IP address
  - SECLABEL ya is SYSMULTI if the stack is unrestricted or can be anything for a VIPA address
  - SECLABEL ya is equivalent to y if the stack is restricted
- ZONEB is the remote IP address
  - SECLABEL yb must be equivalent to y if the stack is restricted
  - SECLABEL yb must be equivalent to w if the stack is unrestricte
- All other addresses are in the « UNKNOWN » zone
  - Define one security label that has the lowest security level and one category that is not used in any other security labels

Figure 4-5 Restricted and unrestricted stacks

Note on the use of VIPAs

It should be pointed out that the SECLABELs specified for a given stack's IP addresses (including LOOPBACK and IN_ADDRANY) must be the same as the TCPIP job and STACKACCESS profile SECLABELs. The only exception is that VIPAs defined on a SYSMULTI stack may have any valid SECLABEL for that system image.

Exempt users

A SYSMULTI user with UPDATE authority to the EZB.STACKACCESS profile will be exempt from the Network Access Control restriction that all traffic must be with partners that are in security zones with security labels that are equivalent to the stack's security label or the security label associated with the local IP address.

It is recommended that this authority be limited to their usage of the programs that must be exempted. This can be accomplished by first specifying UACC(READ) when defining the STACKACCESS profiles, and then granting conditional update access to each by specifying the following:

```
PERMIT stackaccess_profile_name CLASS(SERVAUTH) ID(*) ACCESS(UPDATE) -
WHEN(PROGRAM(ping,oping,tracerte,otracert,omproute))
```

Note: The WHEN(PROGRAM()) conditional access parameter is not supported on profiles in the SERVAUTH class, except where explicitly stated. PERMITs with WHEN(PROGRAM()) on other profiles might be ignored.

Stack recognition of a multilevel secure environment

What follows is fully available at z/OS 1.6, which was the level we used for this book. We learned that the z/OS 1.5 stack is much more restrictive about the sequence of events and does not tolerate some dynamic changes very well (like cycling between SETROPTS NOMLACTIVE and SETROPTS MLACTIVE).

You can activate the SAF SECLABEL class and define security labels on SERVAUTH profiles. This causes the security server to enforce Mandatory Access Control policies for those resources without fully activating a multilevel secure environment.

The z/OS Communications Server stack does not perform its extra Mandatory Access Control policy enforcement until you issue the RACF command SETROPTS MLACTIVE. When a NETACCESS statement is encountered in TCPIP profile processing and MLACTIVE has been set, the stack activates extra Mandatory Access Control policy enforcement in both restricted and unrestricted stacks, as follows:

- New sockets are allowed only if a STACKACCESS profile covers this stack.
- Network access is allowed only to IP addresses that are mapped into network security zones covered by NETACCESS profiles.
- Restricted stacks do not normally allow SYSMULTI tasks to have network access to security zones with security labels that are not equivalent to the stack's security label.
- Unrestricted stacks transmit packet labels both internally and externally to enable an extra Mandatory Access Control check, between the sending task's security label and the receiving task's security label, when both IP addresses are in security zones with a SYSMULTI security label.
- Distributing stacks consider security labels in choosing target applications. In-stack TN3270E servers consider security labels in mapping connections to LU names.
- Internal configuration consistency checks are performed whenever PROFILE.TCPIP or certain SERVAUTH class profile changes are made.

4.3 Setting up MLS for z/OS TCP/IP communications

In this section we provide an example of an MLS network setup for TCP communications.

4.3.1 Our test configuration

Out test configuration is shown in Figure 4-6 on page 96. An MLS z/OS system provides FTP services to another z/OS host, configured as an SLS system, with IP address 10.6.6.81 and workstations located anywhere else in the 10.0.0.0 network. The definitions of the security zone in the TCPIP.PROFILE data set are as shown in Figure 4-7 on page 96. Note that the The z/OS MLS system stack runs with the user ID TCPIP, which is associated with a security label SYSMULTI (that is, an unrestricted stack).

The FTP server has been set up with PORTOFENTRY4 SERVAUTH, and is invoked to transfer the data from/to the data set called MLS4.TEST.L3C. This data set is protected with the arbitrary L3C security label, the combination of security level 3 and category C. The FTP service is run, after proper user authentication, under the user ID MLS4, which is itself defined with the default security label L3C and is permitted to both security labels L3C and L4C. L4C, the combination of security level 4 and category C, dominates L3C.

The SERVAUTH profiles matching this configuration are defined as indicated in Figure 4-8 on page 96. Requests sent from the client with IP address 10.6.6.81 are associated with the port of entry EZB.NETACCESS.TC7.TCPIP.SC60 with the security label L3C, whereas requests sent from other clients in the 10.0.0.0 network come through the port of entry EZB.NETACCESS.TC7.TCPIP.OTHERS with security label L4C.

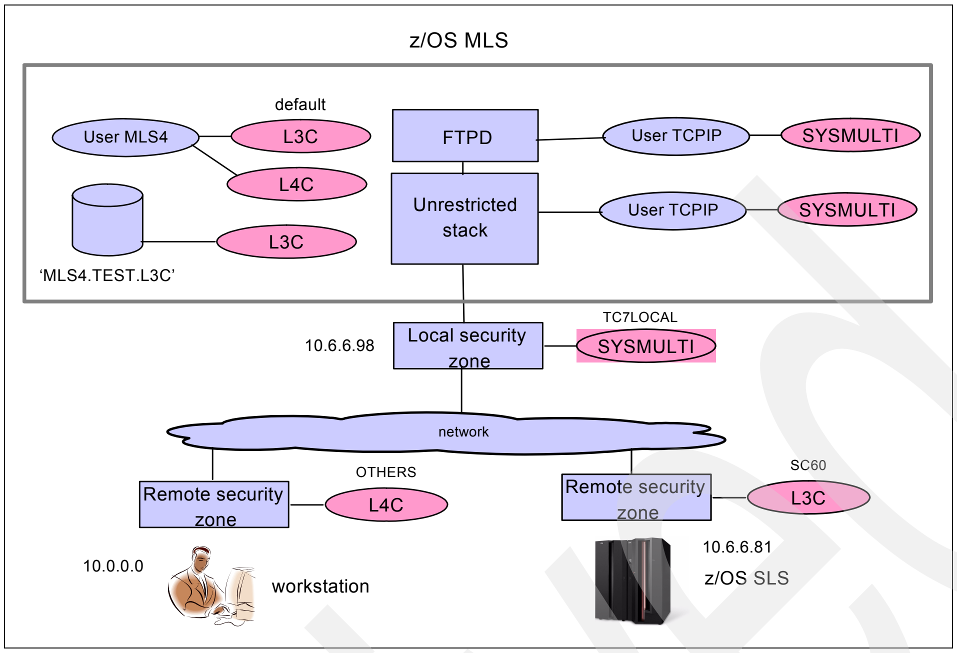


Figure 4-6 Our test configuration

```
NETACCESS INB OUTB
; Network                    SAF
 0.0.0.0/32                  STAKZONE ; INADDR_ANY
 127.0.0.1/8                 STAKZONE ; INADDR_LOOPBACK
 10.6.6.98/32                TC7LOCAL ; system TC7local IP address
 10.6.6.81/32                SC60 ; system SC60 IP address
10.0.0.0/8 OTHERS ; other clients
 DEFAULT                     DEFAULT ; not 10.0.0.0 network
ENDNETACCESS
```

Figure 4-7 Security zones definition

```
RDEF SERVAUTH EZB.STACKACCESS.TC7.TCPIP UACC(READ) SECLABEL(SYSMULTI)
RDEF SERVAUTH EZB.NETACCESS.TC7.TCPIP.STAKZONE UACC(READ) SECLABEL(SYSMULTI)
RDEF SERVAUTH EZB.NETACCESS.TC7.TCPIP.TC7LOCAL UACC(READ) SECLABEL(SYSMULTI)
RDEF SERVAUTH EZB.NETACCESS.TC7.TCPIP.SC60 UACC(READ) SECLABEL(L3C)
RDEF SERVAUTH EZB.NETACCESS.TC7.TCPIP.OTHERS UACC(READ) SECLABEL(L4C)
RDEF SERVAUTH EZB.NETACCESS.TC7.TCPIP.DEFAULT UACC(READ) SECLABEL(SYSNONE)
```

Figure 4-8 SERVAUTH profiles

4.3.2 Our test

In this section we discuss our test.

MLS enforcement

MLS enforcement was turned on with the following RACF commands:

```
SETR CLASSACT(SECLABEL)
SETR MLACTIVE(WARNING)
SETR MLS(FAILURE)
```

Stack recognition of the MLS environment

When running on an MLACTIVE system, the stack performs a Multilevel Security consistency check in any of the following situations:

- After initial profile processing
- After every VARY TCPIP,OBEYFILE command
- After SYSPLEX dynamic VIPA changes
- When it receives an ENF signal from RACF indicating a RACLIST REFRESH was done for the SERVAUTH or SECLABEL class

Figure 4-9 is an example of a failing MLS check. In this case the NETACCESS statements and profiles' setup were correct; however, the stack was started before we turned MLACTIVE on. The MLS check was triggered by a RACLIST REFRESH of SERVAUTH after we issued the SETR MLACTIVE. The stack is actually telling us that it did not perform the required MLS checking the last time it started up.

```
EZD1215I MLSCHK STARTED STACK TCPIP USER TCPIP SECLABEL <NONE>
EZD1224I MLSCHK STACK SECLABEL <NONE> IS NOT VALID
EZD1221I MLSCHK STACKACCESS PROFILE HAS WRONG SECLABEL SYSMULTI 45
         RESNM EZB.STACKACCESS.TC7.TCPIP
         PRFNM EZB.STACKACCESS.TC7.TCPIP
EZD1223I MLSCHK LOCAL ZONE HAS INCORRECT SECLABEL SYSMULTI 457
         IPADR 10.6.6.98  IF OSA2CC0LNK
         ZONE  TC7LOCAL 10.6.6.98/32
         RESNM EZB.NETACCESS.TC7.TCPIP.TC7LOCAL
         PRFNM EZB.NETACCESS.TC7.TCPIP.TC7LOCAL
EZD1223I MLSCHK LOCAL ZONE HAS INCORRECT SECLABEL SYSMULTI 458
         IPADR 127.0.0.1  IF LOOPBACK
         ZONE  STAKZONE 127.0.0.0/8
         RESNM EZB.NETACCESS.TC7.TCPIP.STAKZONE
         PRFNM EZB.NETACCESS.TC7.TCPIP.STAKZONE
EZD1226I MLSCHK INADDR_ANY 0.0.0.0 HAS INCORRECT SECLABEL SYSMULTI
         ZONE  STAKZONE 0.0.0.0/32
RESNM EZB.NETACCESS.TC7.TCPIP.STAKZONE
         PRFNM EZB.NETACCESS.TC7.TCPIP.STAKZONE
EZD1217I MLSCHK FAILED STACK TCPIP - 5 MESSAGES WRITTEN TO JOBLOG
```

Figure 4-9 Failing the MLS check

Stopping and restarting the stack with MLACTIVE on showed that the MLS check passed successfully, as shown in Figure 4-10 on page 98.

```
$HASP373 TCPIP    STARTED
IEE252I MEMBER CTIEZB00 FOUND IN SYS1.IBM.PARMLIB
IEE252I MEMBER CTIIDS00 FOUND IN SYS1.IBM.PARMLIB
EZZ7450I FFST SUBSYSTEM IS NOT INSTALLED
EZZ0300I OPENED PROFILE FILE DD:PROFILE
EZZ0309I PROFILE PROCESSING BEGINNING FOR DD:PROFILE
EZZ0316I PROFILE PROCESSING COMPLETE FOR FILE DD:PROFILE
EZD1215I MLSCHK STARTED STACK TCPIP USER TCPIP SECLABEL SYSMULTI
EZD1216I MLSCHK SUCCEEDED STACK TCPIP IS MULTILEVEL SECURE
EZZ0641I IP FORWARDING NOFWDMULTIPATH SUPPORT IS ENABLED
EZZ0350I SYSPLEX ROUTING SUPPORT IS ENABLED
EZZ0352I VARIABLE SUBNETTING SUPPORT IS ENABLED FOR OROUTED
...
```

Figure 4-10 Passing the MLS check

The test

In this section we discuss the test.

Reading and writing the data set from the OTHERS security zone

We had a workstation located in the 10.x.x.x subnetwork hosting the FTP client. The FTP requests were therefore identified at the MLS z/OS as coming from the OTHERS security zone, and therefore the POE was tagged with the L4C security label. We authenticated at the FTP server as user MLS4, who is permitted to SECLABEL L4C, and tried to read and write the ‘MLS4.TEST.L3C’ data set, which has itself SECLABEL L3C. The results, as seen from the workstation, are shown in Figure 4-11. The read operation did succeed as the request was running with a security label (L4C) dominating the data set security label (L3C); however, for the same reason and because we had MLS(FAILURE) turned on, we failed the writing, with the system console displaying the RACF message shown in Figure 4-12 on page 99.

```
C:\Documents and Settings\kappeler>ftp 9.12.6.98
Connected to 10.6.6.98.
220-FTPD1 IBM FTP CS V1R6 at TC7.ITSO.IBM.COM, 18:20:
220 Connection will close if idle for more than 5 min
User (10.6.6.98:(none)): mls4
331 Send password please.
Password:
230 MLS4 is logged on.  Working directory is "MLS4.".
ftp> get 'mls4.test.l3c' c:\transfer\test.data
200 Port request OK.
125 Sending data set MLS4.TEST.L3C
250 Transfer completed successfully.
ftp: 25915 bytes received in 4,61Seconds 5,63Kbytes/s
ftp> put' c:\transfer\test.data 'mls4.test.l3c'
Invalid command.
ftp> put c:\transfer\test.data 'mls4.test.l3c'
200 Port request OK.
550 STOR fails: MLS4.TEST.L3C.  User not authorized.
```

Figure 4-11 FTP client in the OTHERS security zone

```
ICH408I USER(MLS4    ) GROUP(SYS1    ) NAME(MLS4
  MLS4.TEST.L3C CL(DATASET ) VOL(TC7TS1)
  INSUFFICIENT SECURITY LABEL AUTHORITY
  ACCESS INTENT(UPDATE )  ACCESS ALLOWED(NONE   )
```

Figure 4-12 No write-down RACF message

Note: We did not use here the REPLYSECURITYLEVEL statement in FTP.DATA. A REPLYSECURITYLEVEL 1 statement would have resulted in hiding IP addresses, host names, port numbers, or server operating system level information in FTP replies.

Reading and writing the data set from the SC60 security zone

When using the FTP client in the z/OS SLS system located at 10.6.6.81, the request is identified as coming from the SC60 security zone with SECLABEL L3C. After proper authentication of the MLS4 user at the FTP server, reading and writing the ‘MLS4.TEST.L3C’ data set succeeds, as the POE is associated to SECLABEL L3C. This is shown in Figure 4-13 on page 100.

```
EZA1450I IBM FTP CS V1R6
EZA1554I Connecting to:   10.6.6.98 port: 21.
220-FTPD1 IBM FTP CS V1R6 at TC7.ITSO.IBM.COM, 18:23:54 on 2005-02-13.
220 Connection will close if idle for more than 5 minutes.
 EZA1459I NAME (10.6.6.98:MLS6):
mls4
 EZA1701I >>> USER mls4
 331 Send password please.
 EZA1789I PASSWORD:


 EZA1701I >>> PASS
 230 MLS4 is logged on.  Working directory is "MLS4.".
 EZA1460I Command:
get 'mls4.test.l3c' 'mls4.test.sc60' (replace
 EZA1701I >>> PORT 9,12,6,81,4,41
 200 Port request OK.
 EZA1701I >>> RETR 'mls4.test.l3c'
 125 Sending data set MLS4.TEST.L3C
 250 Transfer completed successfully.
 EZA1617I 25915 bytes transferred in 0.060 seconds.  Transfer rate 431.92
Kbytes
          /sec.
 EZA1460I Command:
put 'mls4.test.sc60' 'mls4.test.l3c'
EZA1701I >>> SITE VARrecfm LRECL=256 RECFM=VB BLKSIZE=6233
200 SITE command was accepted
EZA1701I >>> PORT 9,12,6,81,4,42
200 Port request OK.
EZA1701I >>> STOR 'mls4.test.l3c'
125 Storing data set MLS4.TEST.L3C
250 Transfer completed successfully.
EZA1617I 25915 bytes transferred in 0.010 seconds.  Transfer rate 2591.50
Kbyte
         s/sec.
EZA1460I Command:
```

Figure 4-13 FTP client in the SC60 security zone

4.4 The big theoretical picture - TCP

In this section we give a detailed explanation of the interactions that occur between a TCP client and server running on z/OS with MLS turned on. We are not developing here the UDP or raw sockets communications in an MLS environment, as it is felt that networking is not the primary topic of this book.

Reminder on TCP communications

Figure 4-14 on page 101 describes the sequential interactions occurring between a client, on the left, and a server communicating via the TCP protocol.

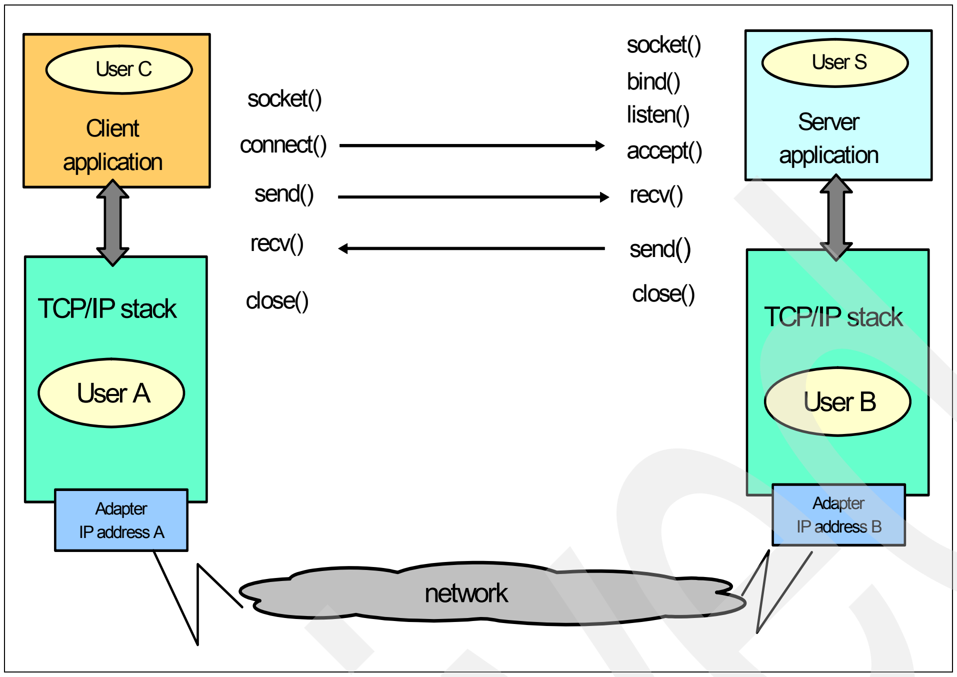


Figure 4-14 TCP/IP client-server interactions

4.4.1 Sequence of events

In this section we discuss the sequence of events.

Socket() and explicit bind()

The first MAC-relevant step in this sequence is the socket() call. The application security label must be equivalent to the security label in the STACKACCESS SERVAUTH profile. Using an unrestricted stack with the SYSMULTI security label would allow all applications to pass this MAC check. The second MAC event is the bind() call. This is described in a generic manner in Figure 4-15 on page 102, where the bind can succeed only if the stack seclabel is equivalent to the server application, and likewise the local IP address maps to a security zone with a seclabel equivalent to the application seclabel.

The server then performs a listen(), which does not have any MAC check, followed by an accept(). Return from the accept() will occur on receipt of a connect() by the client. Note that MAC is checked again on the accept, as explained later. DAC is enforced after successful MAC processing.

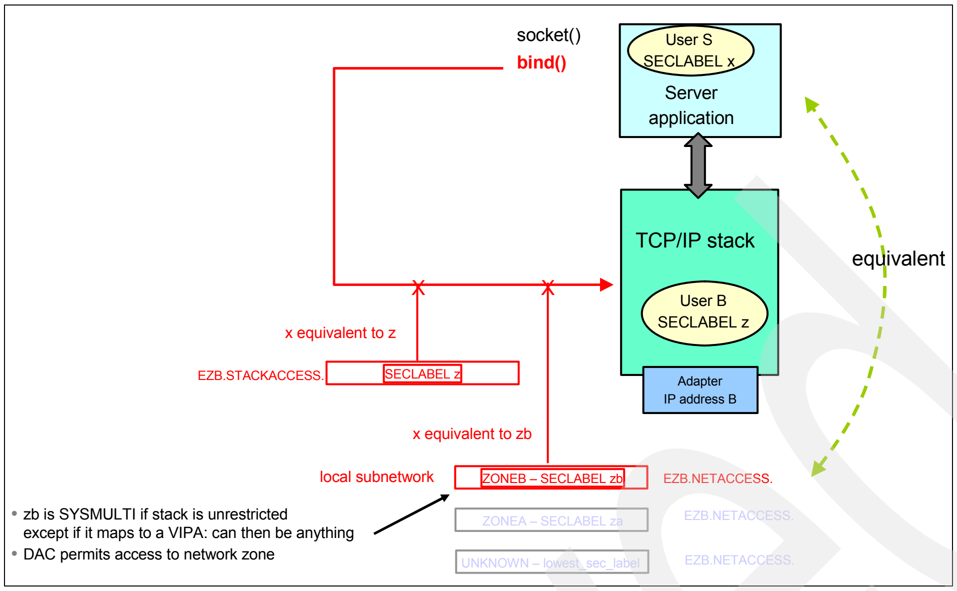


Figure 4-15 Explicit bind()

Implicit bind() and connect()

The next step is for the client, in zone A, to connect to the listening server. In Figure 4-16 we show how the connect() works in a z/OS MLS system. The connect() involves first an implicit bind() that can succeed only if the stack seclabel is equivalent to the client application seclabel, and likewise the local IP address maps to a security zone with an equivalent seclabel equivalent to the application seclabel.

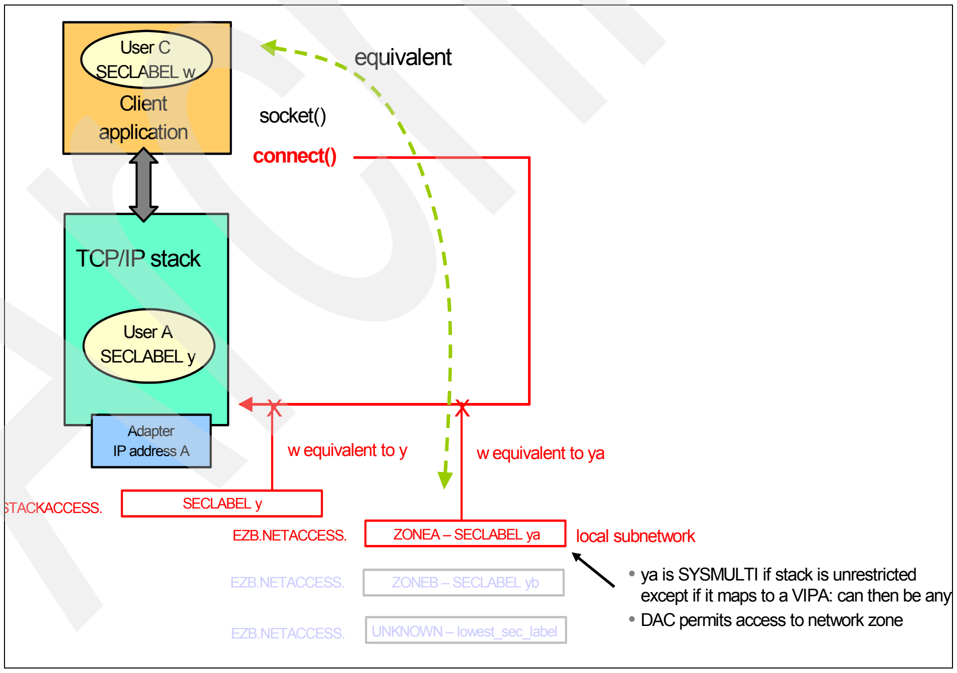


Figure 4-16 TCP connect() implicit bind() 1/2

The actual connect() is performed if the destination zone has a security label equivalent to the client application security label, as shown in Figure 4-17 on page 103. The datagrams to be exchanged must be associated to a security label so that MAC can be applied at the destination of the packet. The possible options to give the packet a security label are shown in Figure 4-18.

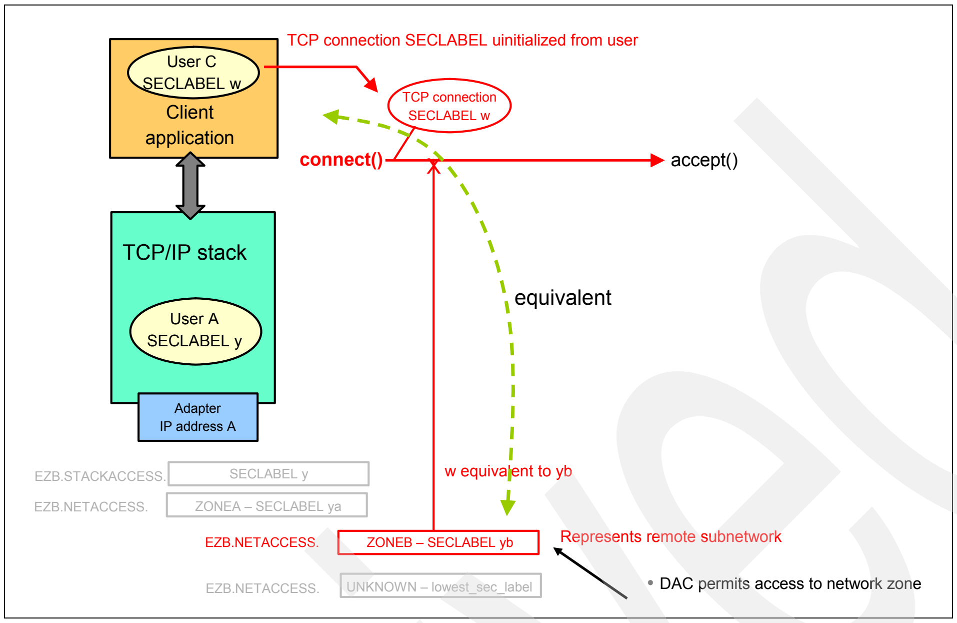


Figure 4-17 TCP connect()

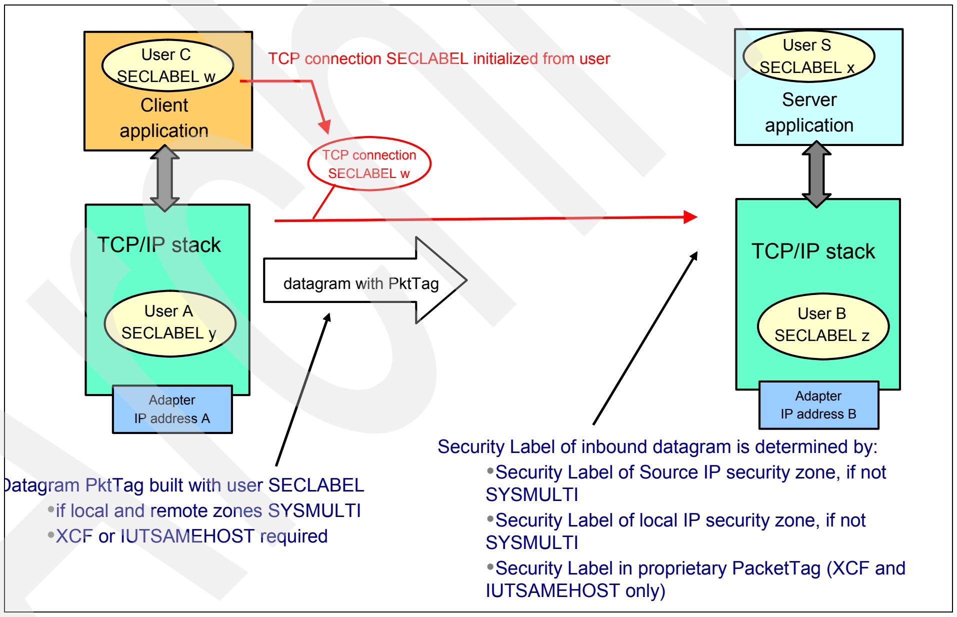

User C
SECLABEL w
Client application
TCP connection SECLABEL initialized from user
User S
SECLABEL x
Server application
TCP connection SECLABEL w
TCP/IP stack
datagram with PktTag
TCP/IP stack
User A
SECLABEL y
User B
SECLABEL z
Adapter
IP address A
Adapter
IP address B
Security Label of inbound datagram is determined by:
•Security Label of Source IP security zone, if not SYSMULTI
•Security Label of local IP security zone, if not SYSMULTI
•Security Label in proprietary PacketTag (XCF and IUTSAMEHOST only)
Datagram PktTag built with user SECLABEL
•if local and remote zones SYSMULTI
•XCF or IUTSAMEHOST required

Figure 4-18 Packet tagging

Return from the accept() function at the server is performed if the connect() packet passes the MAC check, where its security label must be equivalent to the server's application security label, as shown in Figure 4-19 on page 104. Note that the POE security label at the server can be initialized from:

- The packet tag, if present
- The source zone security label
- The destination zone security label

Accept()

During the accept() phase, the connection and POE security label are initialized from the source security zone, the destination security zone, or the packet tag. MAC is checked on accept(), and nonequivalent connections in the backlog are silently reset; the server gets the first equivalent one found. Every subsequent send and receive is checked for equivalence to the partner IP security zone.

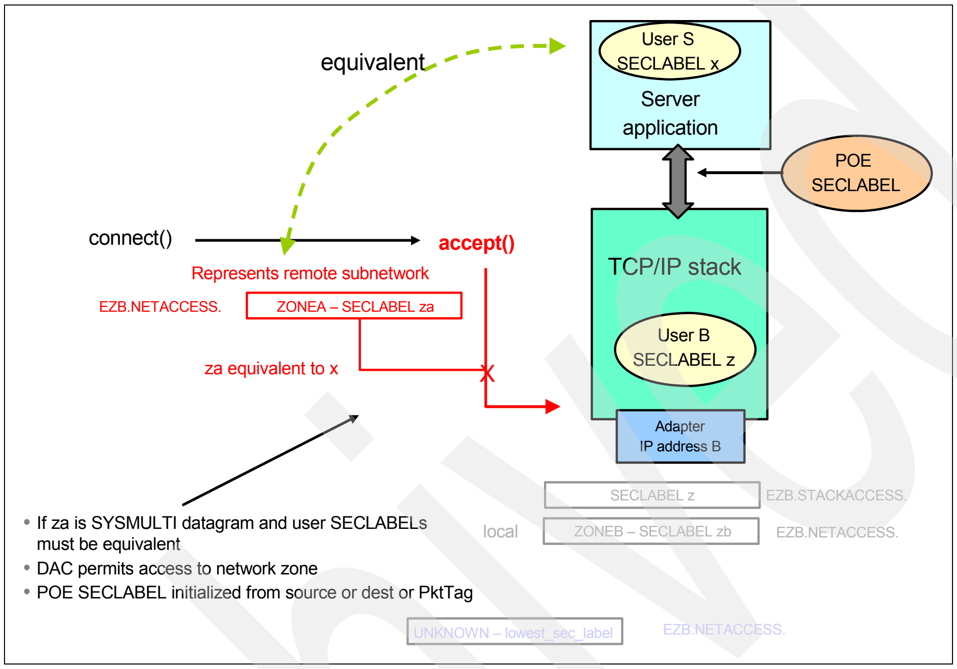


Figure 4-19 Accept()

Send() and receive()

Finally, the data are exchanged via send() and receive() functions, with proper checking of equivalency of security labels, as shown in Figure 4-20.

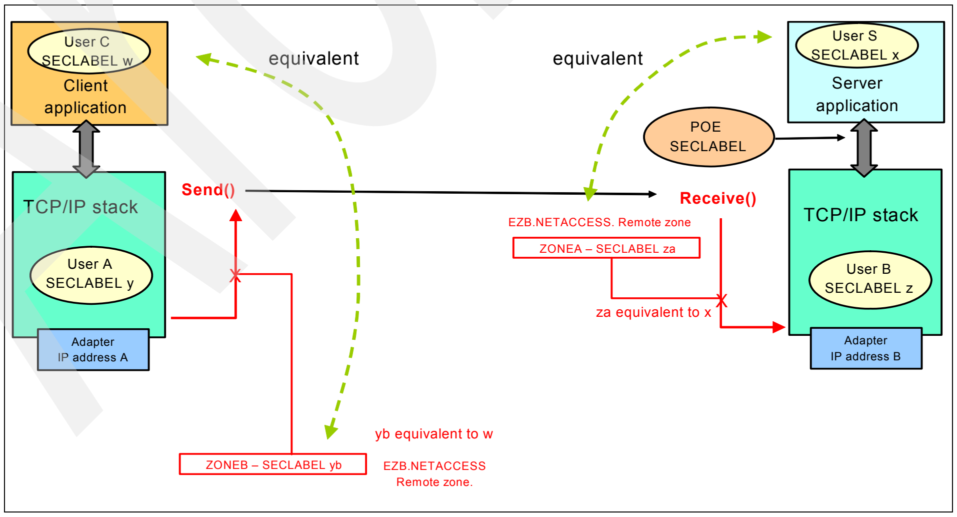


Figure 4-20 Send() and receive()

5

z/OS Integrated Security Services LDAP

This chapter describes and discusses the new features of the z/OS Integrated Security Services LDAP server and client for z/OS1.4 and later.

To build the examples that we are presenting in this chapter, we used an LDAP server at z/OS 1.6, that we installed and configured using the ldapcnf utility.

The LDAP reference books to use are:

- *z/OS Integrated Security Services LDAP Client Programming*, SC24-5924
- *z/OS Integrated Security Services LDAP Server Administration and Use*, SC24-5923

5.1 Some historical data on z/OS LDAP

The LDAP offering on z/OS comprises both an LDAP V3 server and client. The server supports several *backends*, actually connectors, to the data repositories that the z/OS LDAP server externalizes as though they were LDAP directories. The purpose of this externalization is to allow remote data manipulation from LDAP clients, provided the remote user can properly authenticate (authenticated “bind”) to the LDAP server. Figure 5-1 on page 107 gives an overview of the z/OS LDAP implementation.

The LDAP server is a z/OS UNIX application, with run-time parameters provided in the configuration file called slapd.conf. The configuration file allows you to enable the backends:

- The HCD backend allows the z/OS I/O configurations administrator to remotely maintain the IODFs (I/O definition files) from an LDAP client. Note that in this case the activation of an IODF is not permitted to be triggered from an LDAP client.
- The SDBM backend externalizes the USER and GROUP profiles in RACF as though they were entries in a directory tree. The authenticated LDAP user can use LDAP operations to maintain these profiles, as though she were working from a TSO/E session, provided she has, like TSO/E, proper privilege to do so in RACF.
- The RMF™ backend permits transferring RMF data to an LDAP client.
- The TDBM backend is using DB2 tables to store any kind of data that users wish to manipulate. As opposed to the previous backends, the data format and contents are not pre-determined. It is left to the directory designer to use whatever LDAP schema fits the objects to be stored in the DB2 tables.
- The GDBM backend appeared at z/OS 1.5, and is rolled back to z/OS 1.3. It also uses DB2 tables to store data; however, the data format is predetermined, as it must represent a *Change Log notification*. The GDBM backend is discussed in “Change Log, as used by the TDBM backend” on page 110, and also in Chapter 6, “RACF Password Enveloping and z/OS LDAP Change Log” on page 149.

Note in Figure 5-1 on page 107 the client authentication mechanisms supported by the z/OS LDAP server. CRAM-MD5 and DIGEST-MD5 are supported beginning with z/OS 1.4 and are further discussed in this chapter. Also, as per the LDAP terminology, *simple* is here a basic authentication using an LDAP Distinguished Name (DN) and a password (with optional SSL/TLS protection) and *EXTERNAL* refers to client authentication using a client digital certificates with, in that case, a mandatory SSL/TLS protected transport.

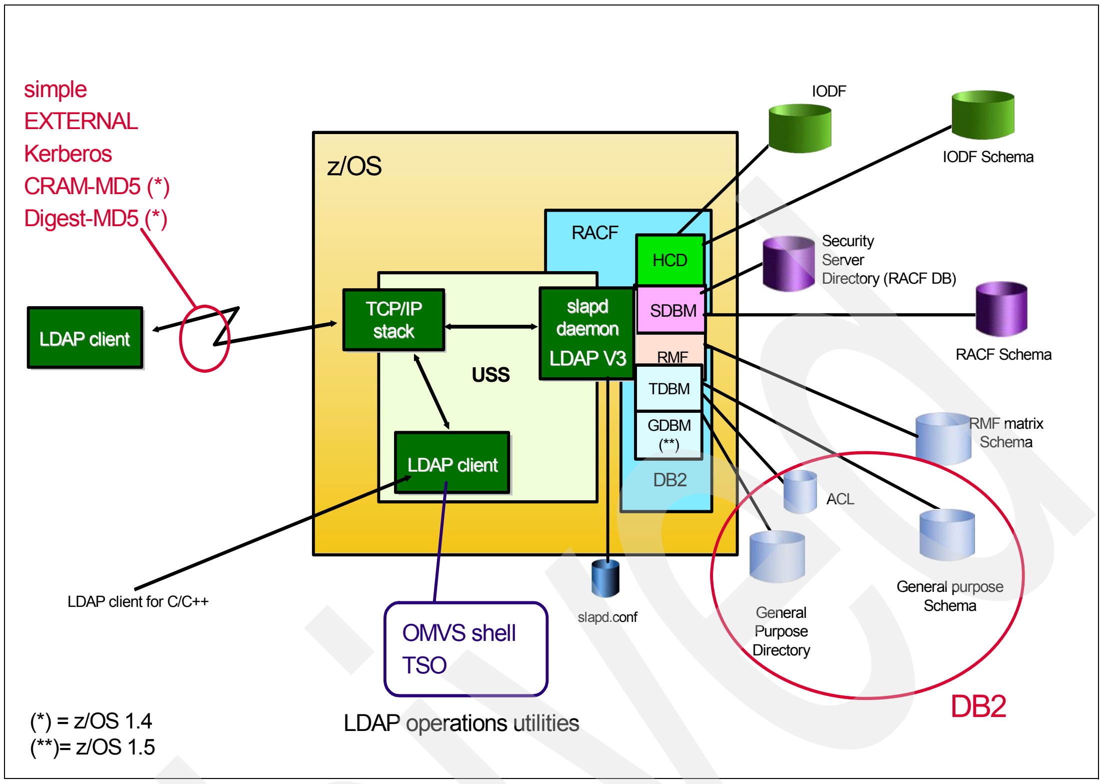


Figure 5-1 z/OS LDAP overview

The z/OS LDAP client

The history of the LDAP client, traced back up to z/OS 1.2, can be summarized as:

- OS/390 2.6: Support of the V3 protocol.
- LDAP client for Java.
  - Available beginning with OS/390 2.7.
  - SSL support added at OS/390 2.8.
  - Removed at z/OS 1.4 - The recommendation is to use IBM's distribution of the Sun support instead.
- The z/OS LDAP client is socksified (SOCKS V4) at OS/390 2.10.
- In z/OS 1.2, Kerberos authentication was made available, in addition to the existing simple and EXTERNAL binding mechanisms.

LDAP authentication mechanisms

In this section we review the LDAP authentication mechanisms.

Reminder: LDAP uses the distinguished name (DN) syntax to identify any object in a directory naming space. This pertains to users as well; when identification is required, the user is eventually designated in the LDAP processes with his distinguished name.

These authentication mechanisms categories are used by the z/OS LDAP:

- Simple authentication

This is typically an authenticated bind using a bind DN and password. Note that if using SSL/TLS to protect the communication, the authentication mechanism is still "simple," as SSL/TLS just operates at the transport level.

- Simple Authentication and Security Layer (SASL)

  The Simple Authentication and Security Layer (SASL) [RFC2222] is a method for adding authentication support to connection-based protocols. To use this specification, a protocol includes a command for identifying and authenticating a user to a server and for optionally negotiating a security layer for subsequent protocol interactions. The command has a required argument identifying a SASL mechanism.

  SASL on z/OS is covered in the following sub-categories:

  - EXTERNAL

    In z/OS this is a bind with a client digital certificate carried over the SSL/TLS protocol.

  - GSSAPI

    The Generalized Security Services API is used in z/OS to perform a Kerberos authentication.

  - CRAM-MD5

    This authentication protocol is enhancing the simple password bind from the security standpoint, and is explained in this chapter.

  - DIGEST-MD5

    This authentication protocol is enhancing the simple password bind from the security standpoint, and is explained in this chapter.

5.2 z/OS LDAP enhancements

In this section we discuss z/OS LDAP enhancements.

5.2.1 Logging support

In this section we cover three topics related to the logging facilities delivered in the z/OS LDAP:

- The activity logging, which was added at z/OS 1.4, the purpose of which is to log server activity data that can be exploited for load analysis or capacity planning.
- The Change Log, with its GDBM backend, as implemented in z/OS 1.5, and rolled back to z/OS 1.3 with APAR OA03857.
- The use of the Change Log by the TDBM backend and the GDBM itself, as introduced by z/OS 1.6 (the only Change Log exploiter at z/OS 1.5 was RACF).

Activity logging

Activity logging is specified by:

- The logfile option in the server's file to locate the file or data set receiving the activity data.
- An MVS modify command to the server started task. Alternatively, environment variables can be set up to have logging starting at the same time as the server.

The activity log file contains information about operations handled by the server, client IP addresses, messages generated by the server, and summary statistics. In general, each record consists of a time stamp followed by the record data. A log record is created when an operation begins and, optionally, another record can be created when the operation completes processing.

Here is a copy in Figure 5-2 of the logfile option that we used in our server configuration file. By coding "//" we are specifying an MVS data set. Anything else will be treated as an HFS (or zFS) file. If you do not specify the logfile option, then log data are collected in the default /etc/ldap/gldlog.output file.

```
logfile //'gld.ldap.activity.log'
```

Figure 5-2 Activity logfile entry in the server configuration file

Note: Ensure the logfile directive starts in column one of the server configuration file and that the started server task has UPDATE access to your log file.

Once we updated our configuration file with our logfile data set name and restarted the server, we issued the commands below. The first command is used to capture usage statistics. The log file is updated on an hourly basis with summary records. The second command will make the server logs at the start of an LDAP add, delete, and modify operations:

```
F GLDSRV,APPL=LOG,SUMMARY
F GLDSRV,APPL=LOG,WRITEOPS
```

We then added a person entry to our LDAP Directory, so that we could check what was captured in the log. After we added the entry we had the server flushing the records out to our log file data set with the first command. Then we stop the activity logging in order to free the data set and browse the collected data:

```
F GLDSRV,APPL=LOG,FLUSH
F GLDSRV,APPL=LOG,STOP
```

Here is, in Figure 5-3, what has been recorded as activity pertaining to our adding of a directory entry. What we are showing here is a short excerpt of the many lines found in the log data set.

```
Mon Aug 23 13:56:11 2004 Add: connid = 1, DN = cn=Walt Farrell, c=fr, IP = 9.100
Mon Aug 23 13:56:23 2004 total operations started = 18
Mon Aug 23 13:56:23 2004 total operations completed = 18
Mon Aug 23 13:56:23 2004 total abandons completed = 0
Mon Aug 23 13:56:23 2004 total adds completed = 4
Mon Aug 23 13:56:23 2004 total binds completed = 1
Mon Aug 23 13:56:23 2004 total compares completed = 0
Mon Aug 23 13:56:23 2004 total deletes completed = 1
Mon Aug 23 13:56:23 2004 total extendedops completed = 0
Mon Aug 23 13:56:23 2004 total modifies completed = 0
Mon Aug 23 13:56:23 2004 total modifydns completed = 0
Mon Aug 23 13:56:23 2004 total searches completed = 12
Mon Aug 23 13:56:23 2004 total unbinds completed = 0
Mon Aug 23 13:56:23 2004 total unknown ops completed = 0
Mon Aug 23 13:56:23 2004 total search entries sent = 16
Mon Aug 23 13:56:23 2004 total bytes sent = 1648
```

Figure 5-3 Sample output from WRITEOPS and SUMMARY

Change Log, as used by the TDBM backend

Here we review the Change Log.

GDBM backend configuration

We summarize here the steps to install the GDBM backend, which proved to be a simple operation, well explained in *z/OS Integrated Security Services LDAP Server Administration and Use*, SC24-5923.

1. Update the LDAP server configuration file.
   a. Add the GDBM backend section.
      - Set Change Log size limits.
      - Start change logging.
   b. Add the SDBM backend section.
   c. Add the listen statement to enable PC Callable Support.
2. Create the DB2 database for the Change Log.
   a. Update SPUFI scripts with unique database owner and name.
   b. Run the scripts.

The GDBM backend maintains a directory of Change Log entries; each entry contains information about a single change and can be inspected by an LDAP client. Note that the GDBM naming space suffix is fixed to "cn=changelog".

Recording TDBM backend changes in the Change Log

The z/OS LDAP server can create at z/OS 1.6 a log of the changes that are made to entries in the TDBM backend. This is different from the activity log seen previously, and the change data are made available in the GDBM directory.

Activating the Change Log capability requires the GDBM back end to be configured in the LDAP server configuration file. Change logging is active by default and pertains to any change made to an entry in the TDBM backend.

The change logging function has no special dependencies. It uses its own set of DB2 tables, which have the same structure as the DB2 tables used by the TDBM backend.

In a sysplex, where there are different levels of the LDAP server sharing the same set of DB2 tables, Change Log entries may be recorded for changes to TDBM backend entries made by the z/OS 1.6 LDAP server.

If you are currently using the TDBM backend and the change logging function for the SDBM (RACF) backend, you will see Change Log entries appearing for the TDBM backend as well after upgrading to z/OS 1.6.

As changes are logged by default in the TDBM backend, you will need to update the LDAP server configuration file, in the TDBM backend section, with the following directive in order to disable change logging:

```
changeLoggingParticipant no
```

When we initially configured our z/OS 1.6 LDAP server, we knew we were going to eventually use the Change Log backend (GDBM) and the RACF backend (SDBM) in addition to the TDBM backend.

Now that we are ready to look at the Change Log, it has collected just over 1100 changes to the entries in the TDBM backend. So, before going further, we empty the Change Log and then update the configuration file so that only changes from the TDBM backend are recorded. Here is, in Figure 5-4, the combination of ldapsearch and ldapdelete commands we used to empty the GDBM backend (note that we have been using the LDAP administrator DN and password).

```
ldapsearch -h 10.11.12.13 -p 389 -D cn=admin -w secret -b "cn=changelog" -s one
"(objectclass=*)" dn  > chg1.out
ldapdelete -h 10.11.12.13 -p 389 -D cn=admin -w secret -f chg1.out
```

Figure 5-4 Emptying the GDBM with ldapsearch and ldapdelete commands

We wanted to try change logging just for the TDBM backend. We updated the configuration file, as shown in Figure 5-5, in the GDBM section, to turn off change logging for the Change Log itself and restarted the LDAP server. What we did was confirmed by the messages shown in Figure 5-6 on page 112.

```
#----------------------------------------------------------------------
#
# changeLoggingParticipant <yes|no>
#
# Description:
#   Allows/disallows change logging for changes made to entries in this
#   backend.  If not specified, the default is yes (changes to entries in
#   this backend are logged).
#
#----------------------------------------------------------------------
#changeLoggingParticipant %GDBM_CHANGELOGGINGPARTICIPANT%
changeLoggingParticipant no
```

Figure 5-5 The Change Log configuration file setting

```
GLD0244I Change logging is enabled

        Logging started status (0 = off, 1 = on): 1

        Limit in seconds on age of Change Log entries (0 = no limit): 0

        Limit on the number of Change Log entries (0 = no limit): 0

        Current number of Change Log entries: 0

        First change number in use: 0
        Last change number in use: 0

GLD3152I The backend containing the following suffix is not participating in change
logging: 'CN=CHANGELOG'.

GLD3151I The backend containing the following suffix is participating in change logging:
'c=fr'.
```

Figure 5-6 Display from SDSF log

Note that the backend with the suffix “c=fr” is our TDBM backend.

If we now run the **ldapsearch** command against the cn=changelog suffix, we are getting nothing back.

Now let us try making some changes to the TDBM backend. Let us add an entry, modify the entry, search the entry, and delete the entry. We used the LDAP commands shown in Figure 5-7, and the corresponding LDIF files are described in Figure 5-8, Figure 5-9, and Figure 5-10 on page 113.

```
ldapadd    -h 10.11.12.13 -p 389 -D cn=admin -w secret -f chg3add.ldif
ldapmodify -h 10.11.12.13 -p 389 -D cn=admin -w secret -f chg3mod.ldif
ldapsearch -h 10.11.12.13 -p 389 -D cn=admin -w secret -b "c=fr" -s one
"(objectclass=person)"
ldapdelete -h 10.11.12.13-p 389 -D cn=admin -w secret -f chg3del.ldif
```

Figure 5-7 ldap commands to manage the TDBM backend entries

```
dn: cn=Noel Smith,  c=fr
objectClass: top
objectClass: person
cn: Noel Smith
sn: Smith
userPassword: noel
```

Figure 5-8 Contents of the chg3add.ldif file

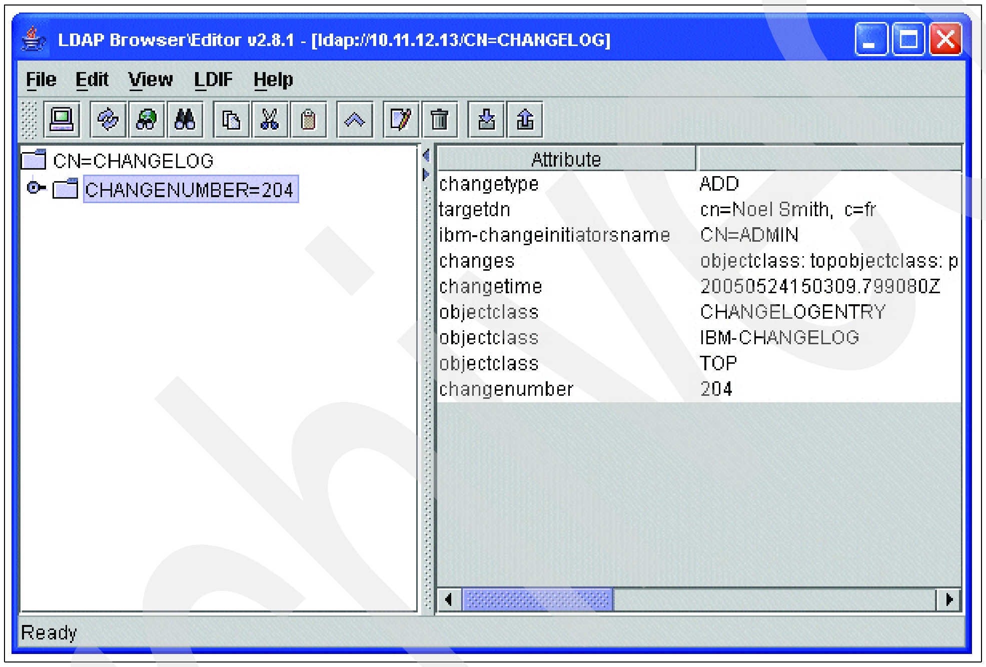
```
dn: cn=Noel Smith, c=fr
changetype: modify
add: telephoneNumber
telephoneNumber: 99 (9)9 99 99 9999
```

Figure 5-9 Contents of the chg3mod.ldif file

```
cn=Noel Smith, c=fr
```

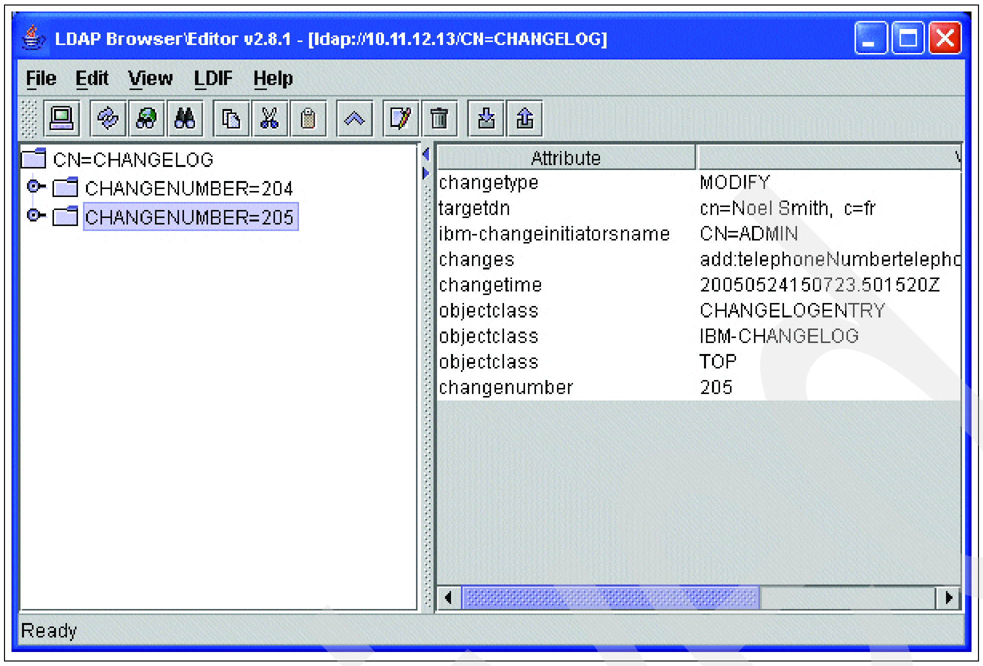
Figure 5-10 Contents of the chg3del.ldif file

The changes can be displayed by running the **`ldapsearch`** command again, but we elected to use an LDAP browser. The Change Log entries for the add, modify, and delete are shown in Figure 5-11, Figure 5-12 on page 114, and Figure 5-13 on page 114. Notice that each entry DN has the CHANGENUMBER attribute, with a unique value. Notice also that the attribute “changetype” in each entry indicates what the nature of the change is in the modified directory, and the “target DN” attribute points at the modified entry in the target directory.


LDAP Browser\Editor v2.8.1 - [ldap://10.11.12.13/CN=CHANGELOG]
File
Edit
View
LDIF
Help
CN=CHANGELOG
CHANGENUMBER=204
Ready

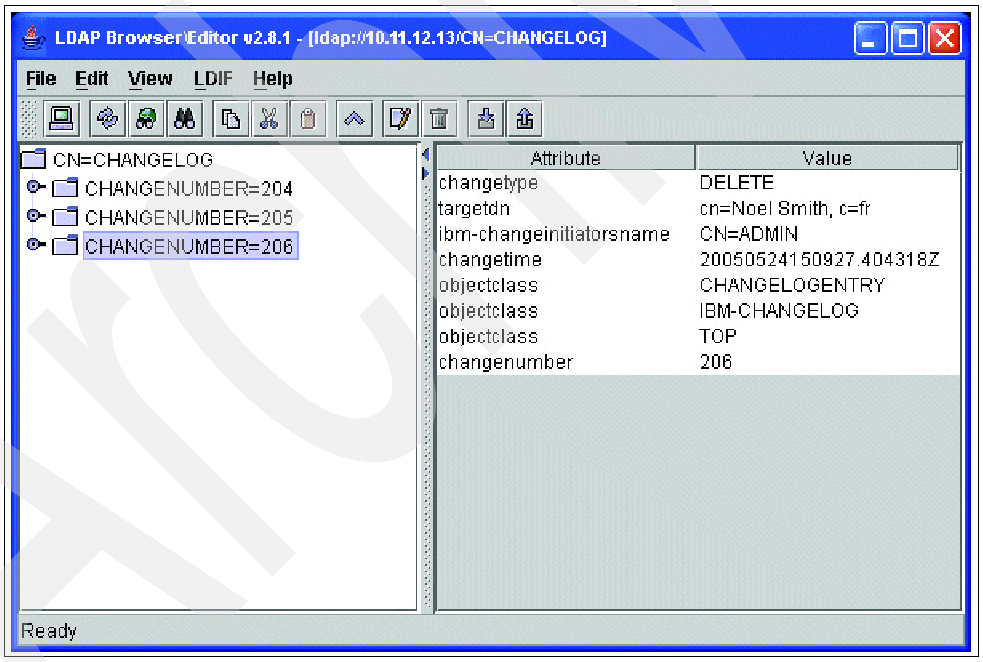
Attribute	
changetype	ADD
targetdn	cn=Noel Smith, c=fr
ibm-changeinitiatorsname	CN=ADMIN
changes	objectclass: topobjectclass: p
changetime	20050524150309.799080Z
objectclass	CHANGELOGENTRY
objectclass	IBM-CHANGELOG
objectclass	TOP
changenumber	204

Figure 5-11 The Change Log entry for an add to the TDBM backend



Attribute	V
changetype	MODIFY
targetdn	cn=Noel Smith, c=fr
ibm-changeinitiatorsname	CN=ADMIN
changes	add:telephoneNumbertelepho
changetime	20050524150723.501520Z
objectclass	CHANGELOGENTRY
objectclass	IBM-CHANGELOG
objectclass	TOP
changenumber	205

Figure 5-12 The Change Log entry for a modify to the TDBM backend



Attribute	Value
changetype	DELETE
targetdn	cn=Noel Smith, c=fr
ibm-changeinitiatorsname	CN=ADMIN
changetime	20050524150927.404318Z
objectclass	CHANGELOGENTRY
objectclass	IBM-CHANGELOG
objectclass	TOP
changenumber	206

Figure 5-13 The Change Log entry for a delete to the TDBM backend

The data stored in the Change Log can be used by applications such as the IBM Tivoli Directory Integrator to synchronize data between application-specific data stores or other directory products.

You should consider adding access controls for the Change Log entry, cn=changelog, so that applications accessing the log data would bind using their own identity, as opposed to using an LDAP administrator identity.

Change Log as used by the GDBM backend

Changes made to the GDBM backend can also be logged in the Change Log.

Note: The only GDBM changes that can create Change Log entries are changes to the GDBM schema, which the client should never have to do; or to the GDBM suffix entry, which the client should do to set an appropriate ACL (you cannot set an ACL on a specific Change Log entry other than the suffix cn=changelog entry).

We amended the configuration file so that the GDBM backend can again participate to change logging. Once we amended our configuration file we stopped and restarted our LDAP server, and we could see the following message displayed:

```
GLD3151I The backend containing the following suffix is participating in change logging:
'CN=CHANGELOG'.
```

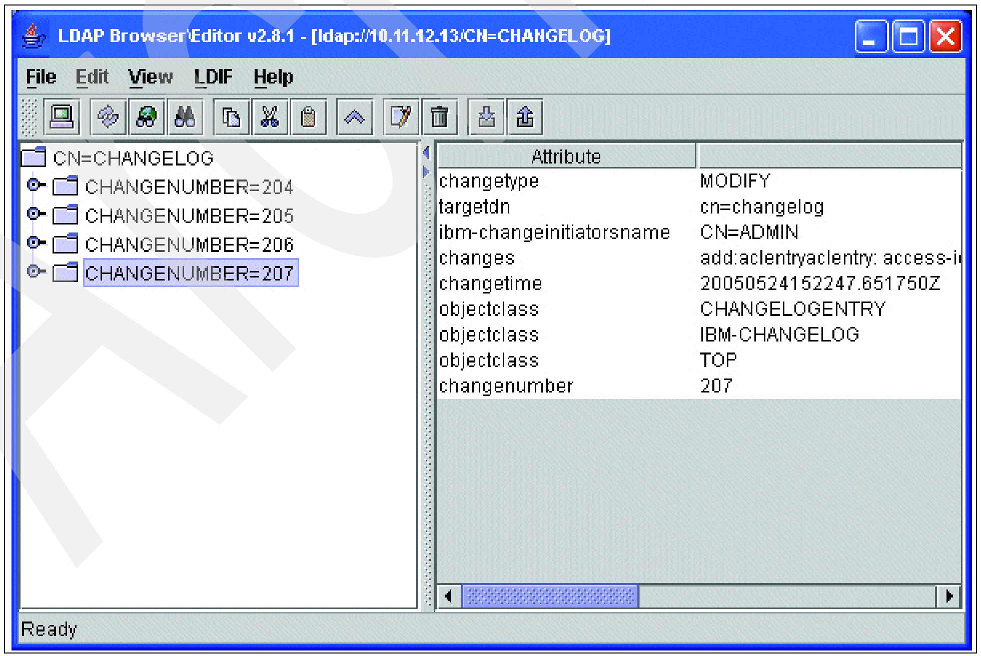
To show how we created an entry into our GDBM backend Change Log we created an LDIF file that adds an aclEntry in the Change Log directory, as shown, and loaded this LDIF file with the command:

```
ldapmodify -h 10.11.12.13-p 389 -D cn=admin -w secret -f chgdbm1.ldif
```

The resulting Change Log entry is shown in Figure 5-15.

```
dn: cn=changelog
changetype: modify
add: aclentry
aclentry: access-id: cn=noel smith, c=fr: normal: rwsc: sensitive: rwsc: critical: rwsc:
  system: rwsc: object: ad
```

Figure 5-14 hgdbm1.ldif - Adding an aclEntry to the Change Log



Attribute	
changetype	MODIFY
targetdn	cn=changelog
ibm-changeinitiatorsname	CN=ADMIN
changes	add:aclentryaclentry: access-i
changetime	20050524152247.651750Z
objectclass	CHANGELOGENTRY
objectclass	IBM-CHANGELOG
objectclass	TOP
changenumber	207

Figure 5-15 The Change Log after a change to the GDBM backend

5.2.2 z/OS LDAP BIND support

The LDAP bind authentication was already achievable on z/OS using a bind DN and password (also called *simple bind*), the password can be protected with SSL at the cost of setting up SSL/TLS in the LDAP client and server. The DIGEST-MD5 and CRAM-MD5 SASL bind mechanisms, which are available today on many LDAP servers and clients, do not pass the credentials in the clear, and do not require any additional products to be installed or configured in order to have this added functionality.

Beginning with z/OS 1.4, the z/OS LDAP server allows clients to authenticate to the server by using CRAM-MD5 (Challenge Response Authentication Mechanism - RFC 2104) and DIGEST-MD5 (RFC 2831) SASL bind mechanisms.

Note that DIGEST-MD5 is a stronger algorithm than CRAM-MD5 from the password secrecy standpoint, and is now required to be implemented in all LDAP V3 compliant servers and clients.

Preamble

CRAM-MD5 and DIGEST-MD5 bind mechanisms are multi-stage binds where the server, when receiving the bind request, sends the client a “challenge,” based on a random number.

The client sends a response back to the server to complete authentication. Instead of a clear password, the client response contains a hash, which is encoded according to the specifications of the DIGEST-MD5 or CRAM-MD5 RFC, and a username.

The username that is specified on the client is also known as the authentication identity, which is used to perform the authentication with the server.

On the z/OS LDAP server, the username is the value of a uid attribute in the TDBM backend entry representing the user. When the server gets the response from the client, the response is parsed and the server calculates its own hash with the password found for the uid attribute value in the backend. If the server hash is equal to the client hash, then the client and the server are authenticated.

Therefore entries in the TDBM backend that represent users are required to contain the following attributes:

- A uid attribute, of which value is the username, or the *authentication identity*, that is to be sent by the LDAP client during the bind.

  It is recommended that this uid value to be unique across the TDBM backend entries, as a bind involves the server side to find the user entry that contains this value of the uid. If its uniqueness cannot be achieved for some reason, then the DN of the entry is also specified in the bind request as an *authorization DN*. If the authentication identity or uid attribute value is present in the authorization DN’s entry, then the bind will be successful, assuming that the password is correct; otherwise it will fail.
- A userpassword attribute, with the recommendation of keeping it encrypted in the entry with the DES 2-way encryption algorithm.

Note: CRAM-MD5 and DIGEST-MD5 binds are not supported with TDBM backend entries that are participating in native authentication.

Other configuration parameter

A *realm name* is expected to be passed by the LDAP server to help create the CRAM-MD5 and DIGEST-MD5 hashes at the client. The value of this parameter gets passed on the initial challenge from the server to the client once it has been decided that a CRAM-MD5 or DIGEST-MD5 bind is desired.

The value of the realm name can be fixed by the digestRealm parameter in the server configuration file. This parameter defaults to the fully qualified host name of the system where the LDAP server is running, assuming that a Domain Name Server (DNS) is available, or it defaults to the name of the host processor.

CRAM-MD5

With the Challenge Response Authentication Mechanism the password is not passed in the clear between the LDAP client and server, making it more difficult for hackers to intercept the password and attack the server. RFC 2195 discusses how to support CRAM-MD5 authentication binds. Figure 5-16 on page 118 is a graphical representation of an LDAP bind using CRAM-MD5.

The LDAP bind proceeds as follow:

1. The client sends over a request to the LDAP Server to do a CRAM-MD5 bind to the TDBM backend.
2. If the LDAP Server supports CRAM-MD5, the LDAP Server creates a challenge and saves it prior to sending it to the client.
3. The client will respond to the LDAP Server with a response to the challenge. The following is the format of the response:

   ```
   " <username> " " <hash>"
   ```

   where the <hash> is calculated using the server challenge and the password is either entered by the user on the client utility or passed in on the API.
4. The server parses the hash and the username passed in by the client.
   - Find the corresponding uid the TDBM backend and its associated password.
   - If the calculated hash is the same as the hash passed in, then the bind is successful, or else it fails.

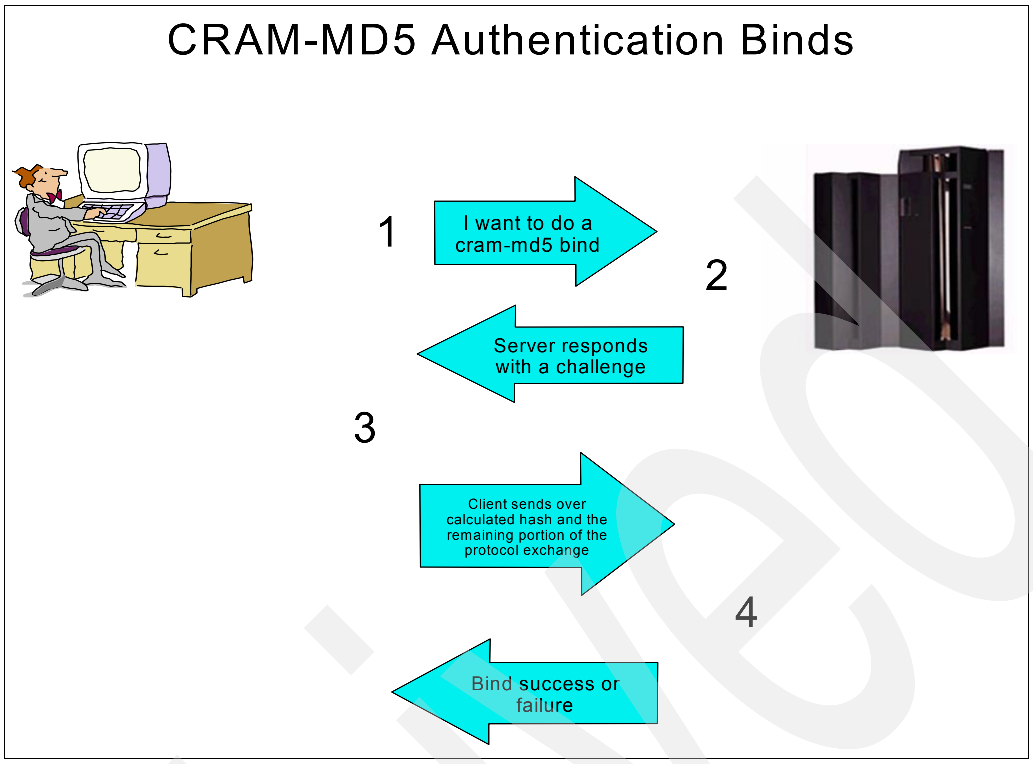

CRAM-MD5 Authentication Binds
1
I want to do a cram-md5 bind
2
Server responds with a challenge
3
Client sends over calculated hash and the remaining portion of the protocol exchange
4
Bind success or failure

Figure 5-16 CRAM-MD5 authentication bind

An example of bind with CRAM-MD5 authentication

To demonstrate the use of CRAM-MD5 and DIGEST-MD5 we have built user entries in our TDBM using the LDIF file shown in Figure 5-17 on page 119. We are then using the ldapsearch client operation utility. Changes to the LDAP operations utilities in order to support the CRAM-MD5 and DIGEST-MD5 authentication mechanisms are described in “Client utility changes for CRAM-MD5 and DIGEST-MD5 support” on page 121.

```
dn: cn=jon,ou=pssc,o=ibm,c=fr
objectclass: top
objectclass: person
objectclass: eperson
cn: jon
sn: briggs
uid: jon
userpassword: passw0rd

dn: cn=peggy1,ou=pssc,o=ibm,c=fr
objectclass: top
objectclass: person
objectclass: eperson
cn: peggy1
sn: LaBelle
uid: peggy
userpassword: passw1rd

dn: cn=peggy2,ou=pssc,o=ibm,c=fr
objectclass: top
objectclass: person
objectclass: eperson
cn: peggy2
sn: LaBelle
uid: peggy
userpassword: passw2rd
```

Figure 5-17 The user entries to test CRAM-MD5 and DIGEST-MD5

To perform a CRAM-MD5 authentication of user jon we issued the following **ldapsearch** command:

```
ldapsearch -h 10.11.12.13 -m CRAM-MD5 -U jon -w passw0rd -b "ou=pssc,o=ibm,c=fr"
"objectclass=*" dn
```

The LDAP server responds with a successful bind and lists the DNs of all entries under the search base.

Now, sending the same command format to authenticate with the uid peggy is going to fail, as the uid peggy is not unique in our TDBM backend, as shown in Figure 5-17:

```
ldapsearch -h 10.11.12.13 -m CRAM-MD5 -U peggy -w passw1rd -b "ou=pssc,o=ibm,c=fr"
"objectclass=*" dn
```

We are receiving from the LDAP server:

```
ldap_sasl_bind_s: Directory server is unwilling to perform the operation
```

This is the case where we need to use an authorization DN to be specified in the **ldapsearch** request. The command below is going to work:

```
ldapsearch -h 10.11.12.13 -m CRAM-MD5 -D "cn=peggy1,ou=pssc,o=ibm,c=fr" -w passw1rd -b
"ou=pssc,o=ibm,c=fr" "objectclass=*" dn
```

DIGEST-MD5

DIGEST-MD5 follows the same principle as CRAM-MD5, although using a more sophisticated algorithm. A graphical view of the DIGEST-MD5 authentication is given in Figure 5-18 on page 120. The authentication steps are:

1. The client sends over a request to the LDAP Server to do a DIGEST-MD5 bind.

2. If the LDAP Server supports DIGEST-MD5 binds, then the LDAP Server creates a challenge and saves it prior to sending it to the client.
3. The client will respond to the LDAP Server with a response to the challenge with a hash computed using a complex algorithm that takes the components of the server challenge and the password entered by the user on the client utility or API and calculates the RESPHASH (according to RFC 2831).
4. The server parses the response passed from the client. The server finds the username and, if it is specified, the authorization DN in the TDBM backend and its associated password. If the server-calculated RESPHASH is the same as the RESPHASH passed in from the client, the bind is successful; otherwise, it fails.

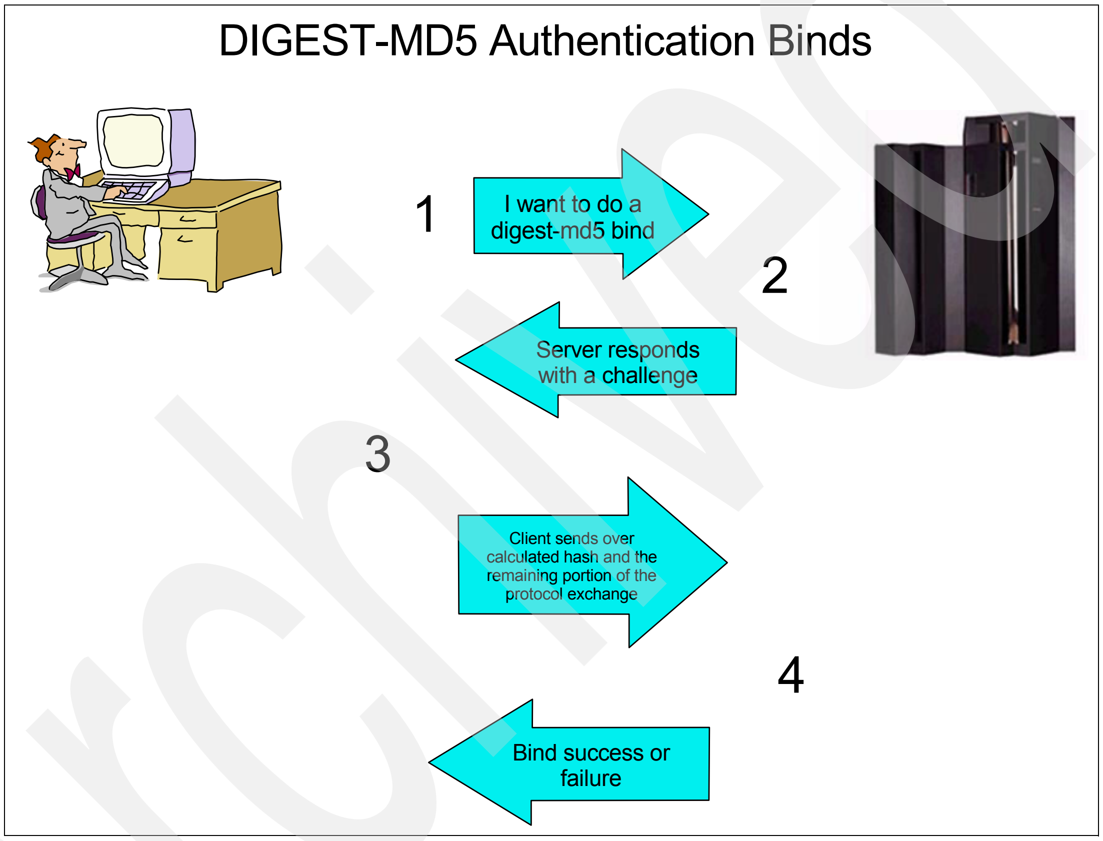

DIGEST-MD5 Authentication Binds
1
I want to do a digest-md5 bind
2
Server responds with a challenge
3
Client sends over calculated hash and the remaining portion of the protocol exchange
4
Bind success or failure

Figure 5-18 DIGEST-MD5 authentication bind

DIGEST-MD5 restrictions on the z/OS LDAP server:

- The options for integrity and encryption protection on DIGEST-MD5 binds are not supported.
- The unspecified user ID form of the authorization identity is not supported; however, the DN version is supported on the z/OS LDAP client and server.
- An authorization DN is used to obtain the authority of another LDAP user after successfully authenticating based upon the authentication identity. The z/OS LDAP server does not support specifying an authorization DN that is different from the authentication identity's DN.
- Subsequent authentication is not supported.

An example of bind with DIGEST-MD5 authentication

We are using the same user entries as for the CRAM-MD5 example. Again we are performing an authenticated ldapsearch for jon, this time using the DIGEST-MD5 authentication mechanism:

```
ldapsearch -h 10.11.12.13 -m DIGEST-MD5 -U jon -w passw0rd -b"ou=pssc,o=ibm,c=fr"
"objectclass=*" dn
```

Doing an authentication using the uid peggy is going to fail because of several LDAP entries containing the same uid value:

```
ldapsearch -h 10.11.12.13 -m DIGEST-MD5 -U peggy -w passw2rd -b "ou=pssc,o=ibm,c=fr"
"objectclass=*" dn
ldap_sasl_bind_s: Directory server is unwilling to perform the operation
```

We need again to use an authorization DN to authenticate as peggy. Note that the DIGEST-MD5 mechanisms still need to have a uid specified along with the authorization DN:

```
ldapsearch -h 10.11.12.13 -m DIGEST-MD5 -U peggy -D “cn=peggy2,ou=pssc,o=ibm,c=fr"-w
passw2rd -b "ou=pssc,o=ibm,c=fr" "objectclass=*" dn
```

Client utility changes for CRAM-MD5 and DIGEST-MD5 support

The following parameters are now available for the bind API:

- -g<realm name> - The DIGEST-MD5 realm that you want to bind with. The realm name is not required for CRAM-MD5 binds. The realm name is used to create the hash value that the protocols need.
- -U<username> - The authentication identity (UID) that you want to bind with, required for DIGEST-MD5.
- m <bind mechanism> - Same meaning as existing -S option bind mechanism.

  The bind mechanisms now supported on -S / -m options are:

  – CRAM-MD5
  – DIGEST-MD5
  – GSSAPI (Kerberos)
  – EXTERNAL (Certificate bind)
  – SIMPLE (default, DN and password)
- Updated existing command line option when CRAM-MD5 or DIGEST-MD5 binds are desired.

  -D <bindDN> - Authorization DN that will be used to perform authorization and access checks; this is optional.

All the client utilities now default to LDAP V3. The client utilities will not follow referrals when doing a CRAM-MD5 or DIGEST-MD5 bind. Note, however, that users can use the LDAP extended re-bind processing API (see below) to allow for referrals in their own applications when doing CRAM-MD5 and DIGEST-MD5 authentication.

TLS support

SSL as such is not provided anymore in new products. It is now replaced by the Transport Layer Security (TLS) protocol, which is the SSL V3 protocol revisited and standardized by the Internet Engineering Task Force (IETF). TLS is not interoperable with SSL; however, TLS provides the capability to revert to SSL when communicating with a non-TLS-capable partner. Note that TLS is sometimes referred to as SSL V3.1

The z/OS LDAP client and server support the TLS protocol beginning with z/OS 1.4.

IPv6 support

IPv6 addressing support has been added to the z/OS LDAP server and client at z/OS 1.6. It mainly deals with any IP address specification in the LDAP server configuration file (for instance, the IP address of a replica server), and in the LDAP client API. An example of IPv4 addressing would be:

```
10.11.12.13:389
```

Whereas an IPV6 address would be:

```
[5f1b:df00:ce3e:e200:800:2078:e3e3]:389
```

The address is enclosed in brackets when there is ambiguity with port specification. IPv6 addresses are always enclosed in brackets when used with an LDAP API or URL.

Asynchronous Kerberos bind

All authentication mechanisms have been supported by the ldap_sasl_bind_s API, which is a synchronous call (that is, the application has to wait for the server to respond before regaining control). However, only simple and EXTERNAL binds were supported on the asynchronous API, ldap_sasl_bind.

With z/OS 1.6, the client application has the option of using Kerberos authentication as an asynchronous bind -ldap_sasl_bind. The client application initiates the bind request and control is returned immediately to the application. The client application calls the ldap_result API to retrieve the response to the bind request.

Extended re-bind processing

In z/OS 1.6 the LDAP client is provided with an enhanced version of the rebind processing called ext_rebind_proc. A re-bind processing is used when the client needs to follow a chain of referrals. When the client binds to the next server in the chain of referrals, the re-bind processing allows the client to provide the necessary bind credentials to the new server. The original rebind_proc only supports simple binds, that is, binds with DN and password, and SASL binds. If the SASL bind was for Kerberos, default credentials (that is, the Kerberos identity used for the initial bind) were used.

The extended re-bind processing, ext_rebind_proc, allows the rebind request to use all of the capability available on a ldap_sasl_bind, for example, Kerberos processing, server and client controls. *z/OS Integrated Security Services LDAP Client Programming*, SC24-5924, contains the function prototypes for the rebind processing and definitions of the parameters.

5.2.3 LDAP access control lists

In this section we discuss LDAP access control lists.

ACL function enhancement

An ACL enhancement is introduced at z/OS 1.4 for the TDBM backend. Prior to this release, access to a specific attribute could be controlled only by the attribute type access-class, that is, *normal*, *sensitive*, *critical*, *restricted*, or *system*, as specified in the schema. With z/OS 1.4, access to a specific attribute can also be explicitly granted or denied at the aclEntry level (and therefore automatically propagated to the entries below). We now illustrate this capability.

When the server starts, the following message in the started task sysout states that the ACL enhancement available in the LDAP server version being used:

```
GLD3135I Grant/Deny ACL support is enabled below suffixes: "c=fr"
```

In our example we show how to set up and use the enhanced ACL function. The directory information tree we created is shown in Figure 5-19.

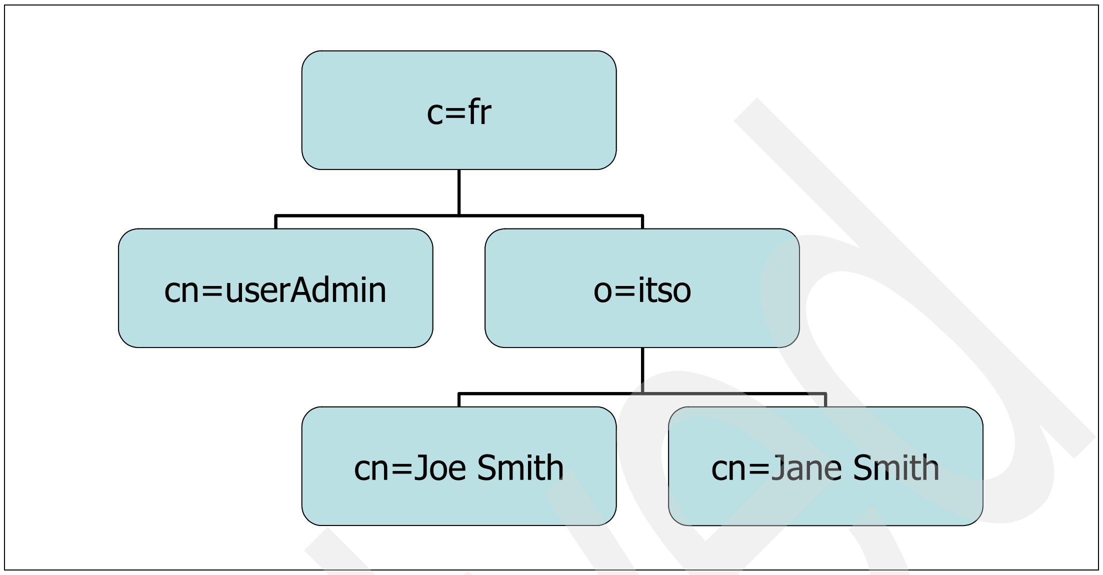


Figure 5-19 Directory information tree used for the enhanced ACL example

The userAdmin entry has been created with the LDIF file shown in Figure 5-20 on page 124. And the user Joe Smith's entry has been created with the LDIF file shown in Figure 5-21 on page 124. The commands used to install these LDIF data were:

```
ldapadd -h 10.11.12.13-p 389 -D cn=admin -w secret -f userAdmin.ldif
```

And:

```
ldapadd -h 10.11.12.13-p 389 -D cn=userAdmin,c=fr -w passw -f joe.ldif
```

However, we failed on this command, with the indication that userAdmin did not have the proper privilege to add an entry in the subtree, which is an expected situation, since initially only the LDAP administrator has add authority:

```
adding new entry cn=Joe Smith, o=itso, c=fr

ldap_add: Insufficient access
ldap_add: additional info: R003057 Access denied because subject does not have add
permission in parent ACL. (aclmgrimpl2.c|1.23|1285), (tdbm_add.c|1.90|745)
```

We had therefore to update the aclEntry via the LDIF file shown in Figure 5-22 on page 124, that we loaded with:

```
ldapmodify -h 10.11.12.13-p 389 -D cn=admin -w secret -f oAcl.ldif
```

We were then able to successfully rerun the add request for the entry Joe Smith, and by the same token we added an entry for Jane Smith.

```
dn: cn=userAdmin,c=fr
objectClass: person
cn: userAdmin
sn: Admin
userPassword: passw

dn: o=itso, c=fr
objectClass: organization
```

Figure 5-20 userAdmin.ldif

```
dn: cn=Joe Smith, o=itso, c=fr
objectClass: person
objectClass: inetOrgPerson
cn: Joe Smith
sn: Smith
uid: MV08
userPassword: secret
homePhone: (845) 555-1212

dn: cn=Jane Smith, o=itso, c=fr
objectClass: person
cn: Jane Smith
sn: Smith
userPassword: secret
```

Figure 5-21 Joe Smith ldif

To fix this error we created oAcl.ldif and gave the user id cn=userAdmin,c=fr authority to add entries under o=itso. We then created the shell script oAcl3.sh, which grants userAdmin authority under o=itso.

```
dn: o=itso, c=fr
changetype: modify
replace: aclEntry
aclEntry: cn=userAdmin, c=fr:normal:rwsc:sensitive:rwsc:critical:rwsc:object:ad
```

Figure 5-22 oAcl. ldif file

The next step was then to perform an ldapsearch, with userAdmin identity, requesting for the found cn and password:

```
ldapsearch -h 10.11.12.13-p 389 -D cn=userAdmin,c=fr -w secret -b "o=itso,c=fr" "cn=smith"
cn userPassword
```

The ldapsearch yields the result shown in Figure 5-23 on page 125.

```
cn=Joe Smith, o=itso, c=fr
cn=Joe Smith
userpassword=secret

cn=Jane Smith, o=itso, c=fr
cn=Jane Smith
userpassword=secret
```

Figure 5-23 ldapsearch result

In our scenario it is not acceptable that userAdmin can see the passwords, but on the other hand we want to have him able to see other attributes with the *critical* access-class.

What we can do, beginning with z/OS 1.4, is to specify a *deny* condition against the password attribute for the userAdmin DN. The LDIF used to do so is shown in Figure 5-24. Note again that the aclEntry value established in the entry “o=itso,c=fr” is automatically propagated to the entries underneath.

```
dn: o=itso, c=fr
changetype: modify
replace: aclEntry
aclEntry: cn=userAdmin,
 c=fr:normal:rwsc:sensitive:rwsc:critical:rwsc:object:ad:at.userPassword:deny:rsc
```

Figure 5-24 oAcl ldif with DENY specified

Now when we run the ldapsearch again under the identity of userAdmin, the results are as shown in Figure 5-25.

```
cn=Joe Smith, o=itso, c=fr
cn=Joe Smith

cn=Jane Smith, o=itso, c=fr
cn=Jane Smith
```

Figure 5-25 Explicit deny in the aclEntry

5.2.4 Enhanced groups support

Prior to z/OS 1.6, the LDAP groups supported by the z/OS LDAP server were *static* groups where members were explicitly listed, and therefore fixed, in the group entry. Groups of groups were not permitted.

At z/OS 1.6 the LDAP server supports the following types of groups:

- Expanded static groups - The group consists of a list of user DNs.
- Dynamic groups - Instead of a list of DNs, membership in a dynamic group is identified by a search filter.
- Groups of groups - The members of the group are groups.

The new group types provide significant administrative advantages. The dynamic group can accurately reflect a changing user population. Users can be added or removed and no change is required to the definition of the group. Support for group of groups is a useful tool for organizing access when different organizations require access to private data as well as data shared across departments.

These groups are supported only in the TDBM backend. They cannot be used with the GDBM or SDBM backends.

In addition to these new types of groups, the ldapsearch or ldapcompare operations for group membership are simplified by specifying as attributes to be returned the ibm-allMembers or the ibm-allGroups attributes.

The ibm-allMembers attribute returns all the members of the group specified as a target of the ldapsearch or ldapcompare.

The ibm-allGroups attribute returns all the groups the target user belongs to.

Attention: The LDAP server checks for group membership at bind time, when the user is authenticated, instead of waiting until an authorization check is performed. This approach results in better performance; however, changes to the bind user's group membership will not be reflected until the next time the user binds to the server.

Expanded static groups

There are five object classes that can be used to create static groups.

- accessRole
- accessGroup (the only object class in the previous static groups implementation)
- ibm-staticGroup
- groupOfNames
- groupOfUniqueNames

The multi-value attribute *member* is used to indicate membership in the group, except for groupOfUniqueNames, where the attribute *uniqueMember* should be used.

Dynamic groups

LDAP entries with the following object classes are considered dynamic groups:.

- ibm-dynamicGroup
- groupOfURLs

The memberURL multi-valued attribute identifies members in the dynamic group. We are providing an example of dynamic group in this section as well.

Groups of groups

Groups of groups, or *nested groups*, make use of the auxiliary class ibm-nestedGroup and the ibm-memberGroup attribute.

Our examples of LDAP groups usage

The Directory Information Tree that we are using is shown in Figure 5-26 on page 127. This directory tree is built, as a start, by using the LDIF file shown in Figure 5-27 on page 128, and loaded with the command:

```
ldapadd -h 10.11.12.13-p 389 -D cn=admin -w secret -f itso.ldif
```

Note: If your z/OS directory server was already installed prior to your upgrade to z/OS 1.6, then you will need first to update the schemas to reflect the new objectclasses and attributes at z/OS 1.6. A sample of the updated schemas can be found in /user/lpp/ldap/etc/schema.user.ldif and /usr/lpp/ldap/etc/schema.IBM.ldif. You will also need to make sure that in your LDAP DB2 database the DB_VERSION has the value ‘3.0’. Refer to the z/OS LDAP server reference book for the SQL statement that needs to be executed to update your DB_VERSION.

The directory now contains examples of the three types of groups. Notice the dynamic group definition syntax. It is of the form:

```
memberURL: ldap:///baseDN??scope?filter
```

Where the memberURL attribute value is a search parameter, stating:

- baseDN: Specifies the location where the search expression starts searching from.
- scope: Must be either *base*, *one*, or *sub*. If not specified, the scope of the search defaults to base.
- filter: Specifies the filter that is to be used on the search expression.

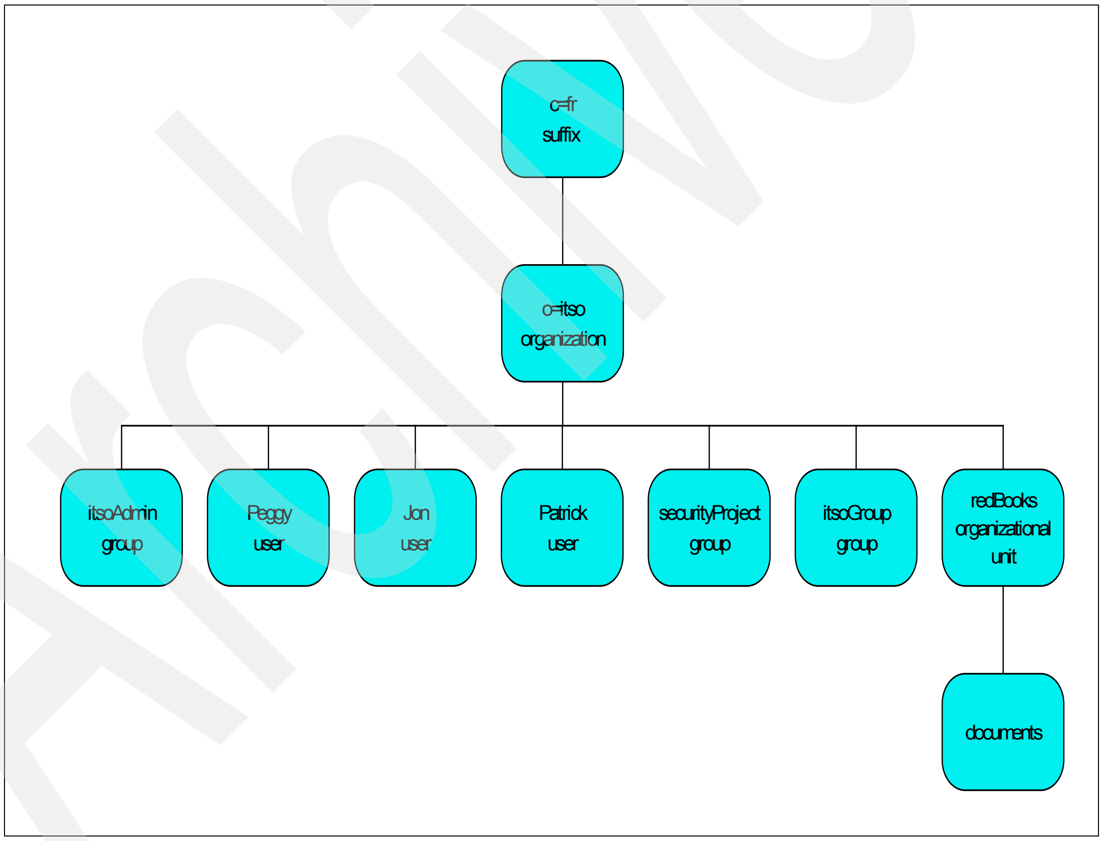


Figure 5-26 Directory information tree

```
# Create the organization
dn: o=itso, c=fr
objectclass: organization
o: itso

# security project
dn: ou=redBooks, o=itso, c=fr
objectclass: organizationalunit
ou: redBooks


dn: cn=Patrick kappeler, o=itso, c=fr
objectclass: person
cn: Patrick kappeler
sn: kappeler
userPassword: patrick

# Create the administration group - A static group
dn: cn=itsoAdmin, o=itso, c=fr
objectclass: accessGroup
cn: itsoAdmin
member: cn=Patrick kappeler, o=itso, c=fr

# Create the security project group - A dynamic group
dn: cn=securityProject, o=itso, c=fr
objectclass: groupOfURLs
cn: securityProject
memberURL: ldap:///o=itso, c=fr??one?objectclass=person

# Create itsoGroup which is a group of groups.
dn: cn=itsoGroup, o=itso, c=fr
objectclass: container
objectclass: ibm-nestedGroup
ibm-memberGroup: cn=itsoAdmin, o=itso, c=fr
ibm-memberGroup: cn=securityProject, o=itso, c=fr
```

Figure 5-27 itso.ldif file

Example 1

We want to create an organization where the administrators are members of the administration group itsoAdmin. So we give the group proper ACL authority in our directory by loading the LDIF file shown in Figure 5-28. It has been loaded with the command:

```
ldapadd -h 10.11.12.13-p 389 -D cn=admin -w secret -f itso.ldif
```

```
dn: o=itso, c=fr
changetype: modify
replace: aclEntry
aclEntry: cn=itsoAdmin, o=itso, c=fr:normal:rwsc:sensitive:rwsc:critical:rwsc:object:ad
  :at.userPassword:deny:rsc
```

Figure 5-28 itsoAdmin group ACL

Patrick is a member of the static group defined, with the DN “cn=itsoAdmin,o=itso,c=fr”, and has therefore the access privilege granted to the group. He can now add the users Jon and Peggy using the LDIF file shown in Figure 5-29.

```
ldapadd -h 10.11.12.13-p 389 -D "cn=Patrick kappeler,o=itso,c=fr" -w patrick -f
project.ldif
```

```
dn: cn=Jon Briggs, o=itso, c=fr
objectclass: person
cn: Jon Briggs
sn: Briggs
userPassword: jon

dn: cn=Peggy LaBelle, o=itso, c=fr
objectclass: person
cn: Peggy LaBelle
sn: LaBelle
userPassword: peggy
```

Figure 5-29 project.ldif

Patrick can now check out the groups members with a search for the ibm-allMembers attribute, as shown in Figure 5-30. The results of the ldapsearch against the static group (itsoAdmin), the dynamic group (securityProject), and the nested groups in itsoGroup are shown in Figure 5-31.

```
ldapsearch -h 10.11.12.13-p 389 -D 'cn=Patrick kappeler,o=itso,c=fr' -w patrick -b
“cn=itsoAdmin,o-itso,c-fr” “objectclass=*” ibm-allMembers
ldapsearch -h 10.11.12.13-p 389 -D 'cn=Patrick kappeler,o=itso,c=fr' -w patrick -b
“cn=securityProject,o=itso,c=fr” “objectclass=*” ibm-allMembers
ldapsearch -h 10.11.12.13-p 389 -D 'cn=Patrick kappeler,o=itso,c=fr' -w patrick -b
“cn=itsoGroup,o=itso,c=fr” “objectclass=*” ibm-allMembers
```

Figure 5-30 groupSearch.sh

```
cn=itsoAdmin, o=itso, c=fr
ibm-allmembers=cn=Patrick kappeler, o=itso, c=fr
cn=securityProject, o=itso, c=fr
ibm-allmembers=cn=Patrick kappeler, o=itso, c=fr
ibm-allmembers=cn=Jon Briggs, o=itso, c=fr
ibm-allmembers=cn=Peggy LaBelle, o=itso, c=fr
cn=itsoGroup, o=itso, c=fr
ibm-allmembers=cn=Patrick kappeler, o=itso, c=fr
ibm-allmembers=cn=Jon Briggs, o=itso, c=fr
ibm-allmembers=cn=Peggy LaBelle, o=itso, c=fr
```

Figure 5-31 Output from search

Now the LDAP Administrator gives members of the itsoGroup rights to operate on entries located in the ou=redBooks sub-tree. This is done by loading the LDIF file shown in Figure 5-32 on page 130.

```
dn: ou=redBooks, o=itso, c=fr
changetype: modify
add: aclEntry
aclEntry:
group:cn=itsoGroup,o=itso,c=fr:object:ad:normal:rwsc:sensitive:rwsc:critical:rwsc
```

Figure 5-32 redbookAcl.ldif

From now on, Peggy and Jon, who are implicit members of the dynamic group itsoGroup can add entries such as the one shown in Figure 5-33.

.

```
# new book
dn: cn=z/OS Security Enhancements, ou=redBooks, o=itso, c=fr
objectclass: document
documentidentifier: SG24-xxxx
documentAuthorCommonName: Jon Briggs
cn: z/OS Security Enhancements
```

Figure 5-33 Adding object in the ou=redBooks sub-tree

5.3 Changes to LDAP operations

In this section we review changes to LDAP operations.

5.3.1 Entry UUID

As part of readying LDAP for Enterprise Identity Mapping, the LDAP server now generates a Universal Unique Identifier (UUID) in each entry in the TDBM backend that is created or undergoes modifications. A UUID is an identifier that is unique across both space and time. A UUID can be used for multiple purposes, from tagging objects with an extremely short lifetime, to reliably identifying very persistent objects across a network.

In z/OS, the generation of UUIDs by the LDAP server can use a derivation of the system's network adapter MAC address, which is to be stored in the slapd.conf file under the keyword serverEtherAdd. The default is to build a UUID based on the CPU model and serial numbers. Remember that a value must be specified if more than one LDAP server is running on a physical system, for it must be unique for all LDAP servers. Since each new/modified entry now contains its UUID, entries may be found by searching the Directory for a matching UUID.

However, some parts of the directory may contain entries, created by previous releases of LDAP. Therefore, the z/OS 1.4 implementation of the LDAP server includes a utility, ldapadduuids, which creates the ibm-entryuuids for existing entries without a UUID.

Note: The utility will never add UUIDs to entries that already have one.

The utility may be run in batch. A sample of the JCL is provided with the LDAP server in the GLDHLQ.SGLDSAMP data set, or ldapadduuids may be run in a shell script. The command line syntax of the utility is similar to ldapsearch.

```
ldapadduuids -D "cn=ldap administrator" -w password -b "o=ibm,c=us" "cn=H*"
```

Figure 5-34 ldapadduuids shell script syntax command

This will add UUIDs to all entries in the tree under o=ibm,c=us that contain a cn attribute value that begins with h or H.

Refer to the chapter “Running and Using the LDAP entry UUID utility” in *z/OS Security Server LDAP Server Administration and Use*, SC24-5923, for the complete syntax of the utility invocation, as well as additional ldapadduuids examples.

To read the UUID on an entry that we created on our z/OS 1.6 system we use the command:

```
ldapsearch -h 10.11.12.13-p 389 -D cn=admin -w secret -b “ou=redBooks,o=itso,c=fr” -s base
“objectclass=*” ibm-entryuuid
```

And we obtain the following answer:

```
ou=redBooks, o=itso, c=fr           ibm-entryuuid=20082000-449B-126C-AFFA-4020641E5223
```

5.3.2 Modify DN

Prior to z/OS 1.4 LDAP, there were few choices when it came to renaming an entry (for example, when a person’s name changed) or for moving a sub-tree to a new location (for example, when a department (OU) was moved under an organization, z/OS LDAP only could rename the leaf nodes).

At z/OS 1.4, the modifyDN operation (ldapmodrdn), as defined in RFC 2251, is fully implemented. It you allows to:

- Rename non-leaf nodes as well as leaf nodes.
- Move sub-tree around as long as its new location is under the same TDBM backend.
- References to the original distinguished name, such as in an aclEntry, can be changed automatically to the new distinguished name.
- The changes in a master server can be replicated to replica z/OS LDAP directory.

A simple Modify DN operation here would be, for example, to change in our previous directory tree the entry DN: ou=redBooks,o=itso,c=fr by DN: ou=redBooks2005,o=itso,c=fr, which is a simple sub-tree renaming, without moving the sub-tree. The command to invoke this modification is:

```
ldapmodrdn -h 10.11.12.13 -p 389 -D cn=admin -w secret "ou=redBooks,o=itso,c=fr"
"ou=redBooks2005"
```

The sub-tree, under ou=redbooks2005, remains the same, except that the last entry we added (refer to Figure 5-33 on page 130) now has a DN:cn=z/OS Security Enhancements, ou=redBooks2005, o=itso, c=fr.

Modifying a DN could easily become a very complex operation to prepare, as all the consequences of moving a sub-tree, for example, have to be understood. Note that there are also performance implications, as it could result in an heavy directory access activity because of the many entries that may need to be changed in the directory, with effects probably seen by the users during the time it takes to complete the modification. It is strongly recommended that you refer to the *z/OS Integrated Security Services LDAP Server Administration and Use*, SC24-5923, and to the entire chapter devoted to the Modify DN operation in order to properly plan and take the measure of the change.

5.3.3 Alias support

At z/OS 1.6, you can create an alias distinguished name for another entry in the TDBM backend. An alias can be a shorter, well known, more familiar name for a longer, more complex distinguished name. Not only can the alias be easier for users to type in, but the entry being aliased can move within the naming space, and the alias still remains the same.

Creating an alias consists of adding an entry for the alias. This entry contains the distinguished name of the target, which can be itself any node in the directory. Note, however, that an alias entry itself must be a leaf entry (that is, has no descendant).

In an LDAP operation, a directory entry DN can contain a mixture of actual relative-DNs and aliases.

Figure 5-35 depicts the directory information tree that we implement for our example. This directory tree is built using the LDIF file shown in Figure 5-36.

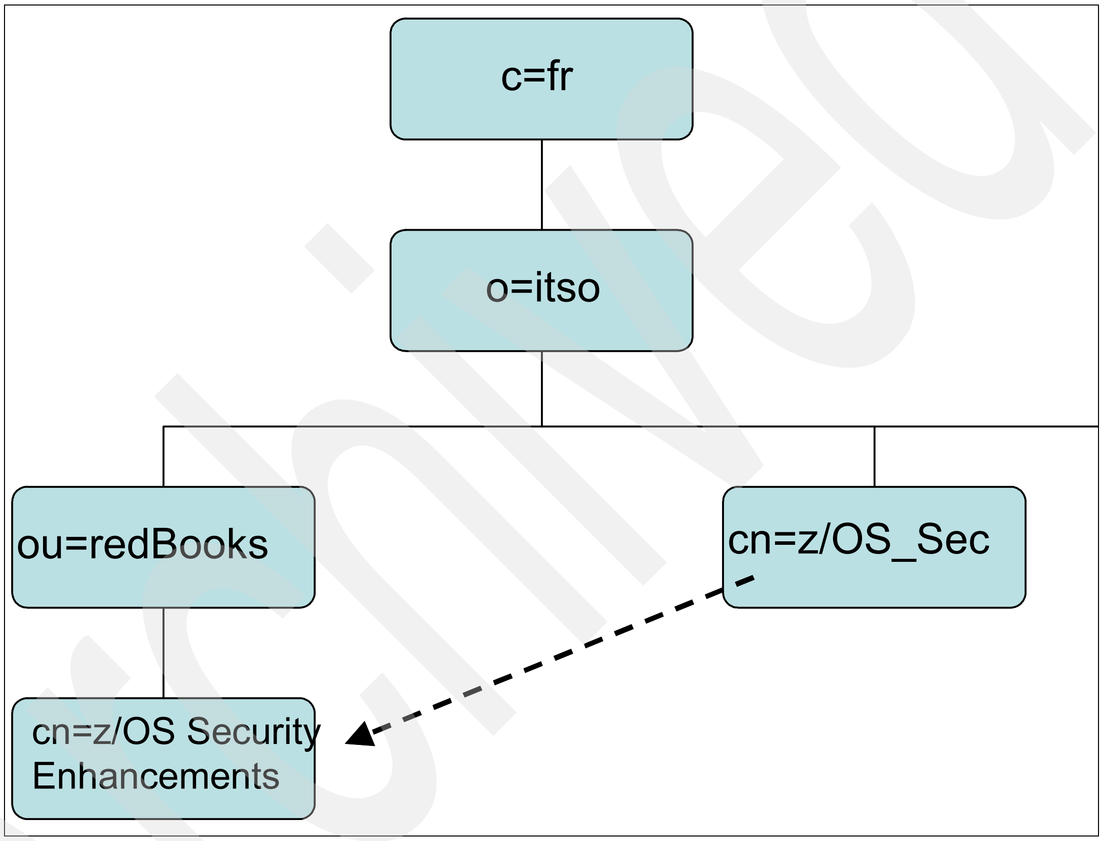


Figure 5-35 Directory information tree for the alias example

```
# Add an alias for the z/OS Security Enhancements book
dn: cn=z/OS_Sec, o=itso, c=fr
objectclass: document
objectclass: aliasObject
cn: z/OS_Sec
documentidentifier: SG24-xxxx
aliasedobjectname: cn=z/OS Security Enhancements, ou=redBooks, o=itso, c=fr
```

Figure 5-36 alias.ldif file

In this example the entry “cn=z/OS Security Enhancements,ou=redBooks, o=itso,c=fr” is aliased by the entry “cn=z/OS_Sec,o=itso,c=fr”—the intent being to have a shorter DN available to get to the book entry.

Note: We chose for structural object class in the alias entry above the object class "document." This forced us to add the documentidentifier attribute, somehow duplicating the aliased entry.

Actually, the alias entry does not have to be the same type of entry as the entry being pointed to. You can choose a simple structural object class such as "container" (this object class just provides cn for the entry's RDN) and then add "aliasObject" to make it an alias. Alternatively, you can use "alias" as your structural object class and then add "extensibleObject" to support any RDN you want.

For the aliasing to work during a search operation, the LDAP operation using an alias as a target must proceed with "dereferencing" the entries it goes through during the search (that is, it does actually inspect the entries to find out if they contain the aliasedObjectName attribute and amends its search accordingly). Note that no access control (ACL) permissions are checked when dereferencing an alias. Normal access control is performed against the dereferenced entries that match the search filter. Thus a search can dereference aliases even though the requestor might not have any permissions to those alias entries.

Dereferencing is an option to specify for a search operation; the default is no-dereferencing.

Note: Alias dereferencing can only be performed during search operations. In all other operations (such as bind, add, modify, delete, compare, and rename), an alias entry is always processed like a normal entry, without dereferencing.

Recommendations

The use of aliases with dereferencing can have a heavy impact on search performance, so use them sparingly. It is better setting up aliases for sub-trees than for leaf nodes. Not only is performance degraded, but aliases also increase the size of the directory. Note that some clients default to using dereferencing so you may need to take special action. The JAVA LDAP interface, the JNDI, has dereferencing as the default, while the z/OS LDAP client does not dereference by default.

Server configuration requirements

There are no hardware, software, or installation requirements to use this function, it works with the z/OS LDAP server TDBM backend. If you are using the z/OS LDAP sysplex support, it is recommended but not required that all of the machines in the sysplex should be at 1.6. The dereferencing flag is not recognized by downlevel versions of the LDAP server so the search results could vary considerably between members.

Before using this function check the version of the LDAP client search to identify what options are used and make adjustments to your programs and procedures as needed.

Alias example

Dereferencing has some fine tuning options that are described in the z/OS LDAP reference books. We are showing here two simple options of an ldapsearch that will use the alias created by loading the LDIF file shown in Figure 5-36 on page 132:

```
ldapadd -h 10.11.12.13-p 389 -D cn=admin -w secret -f alias.ldif
```

We can now show some search examples.

ldapsearch without dereferencing

As already mentioned, dereferencing is not active by default in z/OS LDAP. The search command below yields the result shown in Figure 5-37, which is the actual alias entry itself:

```
ldapsearch -h 10.11.12.13-p 389 -D cn=admin -w secret -b "z/OS_Sec, o=itso, c=fr" 
objectclass=*
```

```
cn=z/OS_Sec, o=itso, c=fr
objectclass=document
objectclass=aliasObject
objectclass=PILOTOBJECT
objectclass=TOP
cn=z/OS_Sec
documentidentifier=SG24-xxxx
aliasedobjectname=cn=z/OS Security Enhancements, ou=redBooks, o=itso, c=fr
```

Figure 5-37 ldap search without dereferencing

ldapsearch with dereferencing

We now specify that we want to proceed with dereferencing with the -a parameter in the **ldapsearch** command set at the value *always*:

```
ldapsearch -h 10.11.12.13-p 389 -D cn=admin -w secret -b "z/OS_Sec, o=itso, c=fr" -a always 
objectclass=*
```

This search gives the information shown in Figure 5-38, which are the aliased entry contents.

```
cn=z/OS Security Enhancements, ou=redbooks, o=itso, c=fr
objectclass=document
objectclass=PILOTOBJECT
objectclass=TOP
documentidentifier=SG24-xxxx
cn=z/OS Security Enhancements
documentAuthorCommonName=jon briggs
```

Figure 5-38 ldapsearch with dereferencing

5.3.4 LDAP search performance improvement

The performance of the z/OS LDAP server is dependant on the use of many caches, which are implemented in the server memory. Caching is particularly important when it comes to retrieving data from the DB2 tables (that is, when using the TDBM or GDBM backends).

At z/OS 1.6 filter and search results caches are added for TDBM and GDBM backends, and improved cache statistics monitoring and tuning capabilities are made available to the users. In prior releases, cache statistics were only available via the LDAP debug facility and at server termination. At z/OS 1.6, cache monitoring and tuning capabilities enable the client to tailor cache sizes to maximize the percent hit rate.

Caching in the z/OS LDAP Server

Here is the list, as of the writing of this book, of the caches that the z/OS LDAP Server uses. In addition to introducing two new caches, z/OS 1.6 also provides statistics information and tuning capabilities for all of them.

ACL source cache	This cache holds information regarding ACL definitions within the database. Retrieval of information from this cache avoids database read operations when resolving access permissions.

DN cache	This cache holds information related to the mapping of distinguished names between their raw form and their canonical form. Retrieval of information from this cache reduces processing required to locate entries in the database.
DN to eid cache	This cache holds information related to the mapping of distinguished names in their canonical form and their entry identifier within the database. Retrieval of information from this cache avoids database read operations when locating entries within the database.
entry owner cache	These are other ACL-related definitions within the database. Retrieval of information from this cache avoids database read operations when resolving access permissions.

New caches at z/OS 1.6

The new caches at z/OS 1.6 are:

entry cache	This cache holds information contained within individual entries in the database. Retrieval of information from this cache avoids database read operations when processing entries within the database.
filter cache	This cache holds information related to the mapping of search request inputs and the result set. Retrieval of information from this cache avoids database read operations when processing search requests.

Example 1 - Obtaining cache statistics with an ldapsearch

Issuing the following ldapsearch returns statistics for the LDAP server and each of the configured backends:

```
ldapsearch -h 10.11.12.13-p 389 -D cn=admin -w secret -b "cn=monitor" objectclass=*
```

The results we received by running this command on our server are shown in Figure 5-39 on page 136.

```
cn=monitor
version=z/OS Version 1 Release 6 Integrated Security Services LDAP Serve
livethreads=10
maxconnections=60
totalconnections=45
currentconnections=1
maxreachedconnections=2
opsinitiated=374
opscompleted=373
abandonsrequested=0
abandonscompleted=0
addsrequested=44
addscompleted=44
bindsrequested=45
bindscompleted=45
comparesrequested=0
comparescompleted=0
deletesrequested=59
deletescompleted=59
extopsrequested=0
extopscompleted=0
modifiesrequested=0
modifiescompleted=0
modifydnsrequested=0
modifydnscompleted=0
searchesrequested=182
searchescompleted=181
unbindsrequested=44
unbindscompleted=44
unknownopsrequested=0
unknownopscompleted=0
entriessent=299
bytessent=43316
searchreferencessent=0
currenttime=Thu Aug 26 17:59:17 2004
starttime=Wed Aug 25 18:30:49 2004
resettime=Wed Aug 25 18:30:49 2004
cn=backendgdbm1,cn=monitor
- - - - - - - - - - - - - - - - - -  204 Line(s) not Displayed
```

Figure 5-39 Output from the ldapsearch cn=monitor command

A quick way to scan the status of the caches is to search for the word "percent" in this report generated by the ldapsearch with a base of cn=monitor. The lower the percent hit rate, the less effective the cache is. After spotting a lower cache rate, take a look at the lines around it in the report to get more information, such as which backend the cache is for. Note that the search can be refined to provide data for a specific backend only by using as a search base:

`cn=backend<name>,cn=monitor`

Where *<name>* is the name given to the backend in the *database* directive in the server configuration file. If the backend was not assigned a name in the configuration file, this part of the search base DN is generated from the type of backend and a number (for example, TDBM1) for the first unnamed TDBM backend in the configuration file.

Manually isolating the lines with "percent" for our TDBM backend monitor report yields the result shown in Figure 5-40 on page 137.

```
- - - - - - - - - - - - - - - - - - -  112 Line(s) not Displayed
cn=backendtdbm1,cn=monitor
- - - - - - - - - - - - - - - - - - - - 32 Line(s) not Displayed
acl_source_cache_percent_hit=0.00%
- - - - - - - - - - - - - - - - - - - -  6 Line(s) not Displayed
dn_cache_percent_hit=96.87%
- - - - - - - - - - - - - - - - - - - -  6 Line(s) not Displayed
dn_to_eid_cache_percent_hit=64.35%
- - - - - - - - - - - - - - - - - - - -  6 Line(s) not Displayed
entry_cache_percent_hit=94.44%
- - - - - - - - - - - - - - - - - - - -  6 Line(s) not Displayed
entry_owner_cache_percent_hit=0.00%
- - - - - - - - - - - - - - - - - - - -  6 Line(s) not Displayed
filter_cache_percent_hit=31.71%
- - - - - - - - - - - - - - - - - - - - 63 Line(s) not Displayed
```

Figure 5-40 Cache hit rates fro the TDBM backend

The ACL cache and the entry owner cache have not been used. The dn-to-eid and entry cache have a high hit rate. The filter cache holds information related to the mapping of search request inputs and the result set. If this hit rate is consistent after a reasonable period of time and does not reflect an imbalance in the server workload, it would be reasonable to try updating the cache size to see if the hit rate could be improved. Here is how we went about changing the cache size in the configuration file of our LDAP server.

We can see in the report what are the actual run parameters of the filter cache; these are the lines in the report that refer to filter_cache, as shown in Figure 5-41.

```
filter_cache_size=5000
filter_cache_current=2
filter_cache_hit=52
filter_cache_miss=112
filter_cache_percent_hit=31.71%
filter_cache_refresh=92
filter_cache_refresh_avgsize=1
filter_cache_bypass_limit=100
```

Figure 5-41 Details about the TDBM backend filter cache

From what was said previously, we can decide to increase the size of the filter cache. This is done by modifying the server's configuration file, as shown in Figure 5-42 on page 138. The change is taken into account after stopping and restarting the server.

```
#---------------------------------------------------------------------
#
# filterCacheSize <number>
#
# Description:
#   Specifies the maximum number of filters to store in the Search Filter
#   cache.  This cache holds information related to the mapping of search
#   request inputs and the result set.  Retrieval of information from this
#   cache avoids database read operations when processing search requests.
#   Individual search requests which return more entries than specified in the
#   filterCacheBypassLimit setting are not placed in the cache.  This cache is
#   not allowed in multi-server mode.
#
#---------------------------------------------------------------------
#filterCacheSize %TDBM_FILTERCACHESIZE%
filterCacheSize 10000

#---------------------------------------------------------------------
#
# filterCacheBypasslimit <number>
#
# Description:
#   Specifies the maximum number of returned entries allowed in the result set
#   of any individual search that will be stored in the Search Filter cache.
#   Search filters that match more than this number of entries will not be
#   added to the Search Filter cache.  This option is useful for maintaining
#   the effectiveness of the Search Filter cache and Entry cache.  It can be
#   used to prevent a few search requests with large result sets from
#   dominating the contents of the Entry cache.  This parameter is ignored
#   when the filter cache is not in use.
#
#---------------------------------------------------------------------
#filterCacheBypasslimit %TDBM_FILTERCACHEBYPASSLIMIT%
filterCacheBypasslimit 200
```

Figure 5-42 Filter cache parameters

Example 2 - Getting performance data with the query command

The LDAP server provides a query command that is used to capture various performance information. The output of the query command goes to the system log and the job output. The syntax of the query command for LDAP is:

```
f <ldap_proc>,appl=query,<report>[,reset][,console|,noconsole]
```

Where:

- *reset* is used to reset the statistics shown in the various reports.
- *console|noconsole* indicates the report output should go to the system console as well as the job log. *noconsole* indicates that the report output should only go to the job log. *console* is the default.
- *report* is one or more of:
  - AIO - This report shows the TCP/IP asynchronous socket calls made by the LDAP server and the amount of queuing for incoming requests.
  - DEBUG - This option shows the current debug setting.

– LOCKING - This report shows the lock facility statistics. It includes the number of lock waits, the average lock wait time, and the time threads sleep waiting for certain specific events.

– MONITOR - This report lists various server statistics related to client connections, client requests, and server caching activity.

– THREADS.

– ALL - This option shows all the reports described above.

Figure 5-43 is an example of an AIO report.

```
 Asynchronous I/O Statistics
            ---------------------------
Current Time:        Thu Aug 26 18:11:42 2004
Last Reset Time:     Wed Aug 25 18:30:53 2004
Number of Resets:    0
Service Thread Stack Size=76K   Number of Pools=3
Pool:   0   Threads:    10     Pool:   1   Threads:    10
Pool:   2   Threads:     0
       SRB Accepts :         45         SRB Requests :        334
        SRB Queued :          0          Pct. Queued :          0.0%
         Schedules :          2              Accepts :         46
             Reads :          0               Readvs :          0
             Recvs :        334            Recvfroms :          0
            Writes :          0              Writevs :          0
             Sends :          0              Sendtos :          0
           Cancels :          0        Cancelsockets :          0
GLD0210I Modify command has been processed successfully: QUERY,AIO
```

Figure 5-43 OQuery report for AIO

Figure 5-44 on page 140 and Figure 5-45 on page 140 show, in sequence, a reset of the MONITOR report:

```
f ldapsrv,appl=query,monitor,reset
```

Followed by:

```
f ldapsrv,appl=query,monitor
```

Note that these figures are shortened versions of what you will actually see at the console or the job log.

```
             Monitor Statistics
             ------------------

Server Version:      z/OS Version 1 Release 6
y Services LDAP Server
Current Time:        Wed Sep 08 09:46:20 2004
Start Time:          Tue Sep 07 13:48:35 2004
Last Reset Time:     Tue Sep 07 13:48:35 2004
Number of Resets:    0
Maintenance Mode     OFF

Server Totals:
--------------

Description                Count
---------------------------------
MaxAllowed Connections        60
Total Connections             41
Current Connections            1
MaxReached Connections         2
Search Entries Sent          280
Message Bytes Sent         42715
Search References Sent         0
```

Figure 5-44 Query MONITOR and RESET

```
               Monitor Statistics
             ------------------

Server Version:      z/OS Version 1 Release 6
y Services LDAP Server
Current Time:        Wed Sep 08 09:56:39 2004
Start Time:          Tue Sep 07 13:48:35 2004
Last Reset Time:     Wed Sep 08 09:46:20 2004
Number of Resets:    1
Maintenance Mode     OFF

Server Totals:
--------------

Description                Count
---------------------------------
MaxAllowed Connections        60
Total Connections              0
Current Connections            1
MaxReached Connections         1
Search Entries Sent            0
Message Bytes Sent             0
Search References Sent         0
```

Figure 5-45 Query MONITOR

5.3.5 LDAP persistent search

The existing LDAP search client interface allows an application to perform a synchronous or an asynchronous search. Once the client receives the results, another ldap_search request has to be made in order for more results to be returned.

With persistent search, as delivered in z/OS 1.6, the LDAP client application defines a search filter at the z/OS server just once. The client can then periodically check for results. Whenever an entry is added or renamed into the search scope, or an entry within the search scope is modified or deleted, and the resulting entry matches the persistent search filter, the entry is returned to the client who created the persistent search.

The persistent search can be used as an event notification mechanism for applications, directories, and meta-directories that need to maintain a local copy, a cache, of directory information that has to synchronize changes with another directory.

On the z/OS LDAP server a persistent search can be used for data that reside in the TDBM and GDBM backends only.

Setting up a persistent search of the Change Log in the GDBM back end is the base mechanism for products such the IBM Directory Integrator to be kept aware of changes to the TDBM backend or the RACF directory entries (assuming that TDBM backend or RACF are set up to notify GDBM of the change).

To set up a persistent search, an application should:

1. Create a persistent search server control using the ldap_create_persistentsearch_control client interface. The ldap_insert_control() routine can be used to add the control to a list of controls for input to the ldap search routine.
2. Pass the server control with a normal asynchronous ldap_search_ext request.
3. The application needs to issue ldap_abandon or ldap_unbind to end the search when it is no longer needed.
4. ldap_free needs to be called to release the persistent search control when no longer needed.

Persistent search is not supported by the ldapsearch command.

The ldap client API is documented in *z/OS Integrated Security Services LDAP Client Programming*, SC24-5924. The persistent search capability is documented in the configuration chapter and an appendix of that book.

Setting up the z/OS LDAP server for persistent search

There are no special migration requirements to enable your z/OS LDAP server for persistent searches. The persistent search capability has to be turned on the TDBM or GDBM backends-specific section in the server's configuration file:

```
persistentSearch yes
```

By default, persistent search is disabled. You can confirm that persistentSearch is actually on by an ldapsearch of the server controls:

```
ldapsearch -h 10.11.12.13-p 389 -D cn=admin -w secret -s base -b "" "objectclass=*"
```

Figure 5-46 on page 142 shows the result of this ldapsearch.

```
supportedcontrol=2.16.840.1.113730.3.4.2
supportedcontrol=1.3.18.0.2.10.2
supportedcontrol=1.3.18.0.2.10.10
supportedcontrol=1.3.18.0.2.10.11
supportedcontrol=1.3.18.0.2.10.20
supportedcontrol=2.16.840.1.113730.3.4.3
supportedextension=1.3.6.1.4.1.1466.20037
ibm-supportedcapabilities=1.3.18.0.2.32.19
ibm-supportedcapabilities=1.3.18.0.2.32.3
ibm-supportedcapabilities=1.3.18.0.2.32.31
ibm-supportedcapabilities=1.3.18.0.2.32.7
ibm-supportedcapabilities=1.3.18.0.2.32.33
ibm-supportedcapabilities=1.3.18.0.2.32.34
ibm-supportedcapabilities=1.3.18.0.2.32.30
ibm-supportedcapabilities=1.3.18.0.2.32.28
ibm-supportedcapabilities=1.3.18.0.2.32.24
namingcontexts=CN=CHANGELOG
```

Figure 5-46 Output from the rootDSE search

Note that the LDAP server controls appear in a standardized format as an Object IDentifier (OID) numeric value. The OID for the persistenSearch is 2.16.840.1.113730.3.4.3.

5.3.6 Peer-to-peer replication

Prior to z/OS 1.6, the z/OS LDAP server could participate to a replica network made up of LDAP servers, where one server was designated as the *master* server and the rest of the servers were *replicas* of the master server. All of the replica servers could process read requests, but only the master LDAP server can accept write requests. The master LDAP server is then responsible for propagating the changes to the replicas.

This configuration could be used to make the LDAP directory more available for readers of the directory information. If a replica went down for maintenance, then users could access any of the other directory servers. The one weakness in this configuration is the master server. If it goes down or is unavailable for maintenance, someone has to reconfigure one of the replicas to be the new master.

z/OS LDAP 1.6 adds a new option for replication: Peer-to-peer replication. The z/OS LDAP server can participate to a replication network of LDAP servers where there is more than one server that can accept write requests. They synchronize updates between themselves. The advantage of this structure is that you now have another server where updates can be made. No re-configuration of your servers is required if the main server becomes unavailable. The IBM Directory Services server on other platforms than z/OS also support peer-to-peer replication.

Setting up peer-to-peer replication in the z/OS LDAP server

On z/OS LDAP peer-to-peer replication is only supported with the TDBM backend.

Peer-to-peer replication is enabled by updating the LDAP server's configuration file. The configuration value peerServerDN indicates that this server is a peer server and accepts replication when bound with this DN. While you can also store the password used to access this server in the configuration file with the peerServerPW directive, it is recommended that you create a directory entry with this DN and store the password into it.

Once the peer-to-peer replication is enabled, then you need to create replica entries for each peer server and each read-only server in the replica network in the directory tree. This replica entry contains the replicaBindDN directives that your server needs to know to bind to the other replicas in the network. An example of a replication network is given in Figure 5-47, where serverA and serverB are peer-to-peer, whereas serverC is a replica of serverA and serverB.

Note: It is recommended that every peer-server be a master server for every read-only replica. In the example shown in Figure 5-47, if serverB fails then ServerA can still replicate the changes to serverC.

The design of peer replication needed to prevent a change from being propagated multiple times to each server. A peer server assumes that any change it receives when bound with a peerServerDN has already been replicated to the other servers in the configuration.

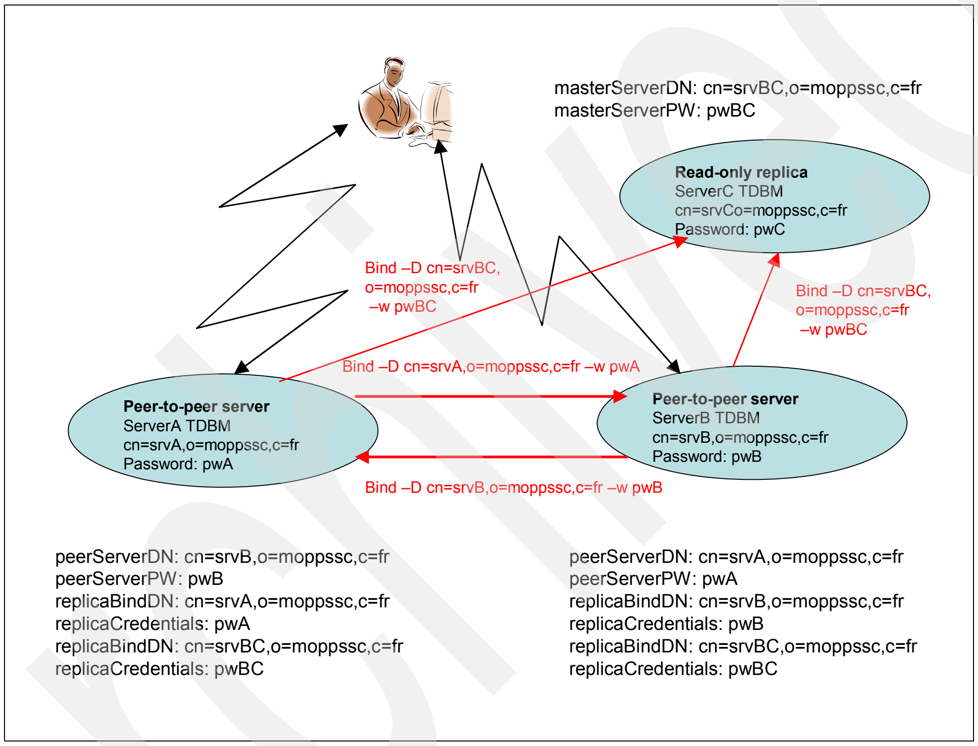

masterServerDN: cn=srvBC,o=moppssc,c=fr
masterServerPW: pwBC
Read-only replica
ServerC TDBM
cn=srvCo=moppssc,c=fr
Password: pwC
Bind –D cn=srvBC, o=moppssc,c=fr –w pwBC
Bind –D cn=srvBC, o=moppssc,c=fr –w pwBC
Bind –D cn=srvA,o=moppssc,c=fr –w pwA
Peer-to-peer server
ServerA TDBM
cn=srvA,o=moppssc,c=fr
Password: pwA
Peer-to-peer server
ServerB TDBM
cn=srvB,o=moppssc,c=fr
Password: pwB
Bind –D cn=srvB,o=moppssc,c=fr –w pwB
peerServerDN: cn=srvB,o=moppssc,c=fr
peerServerPW: pwB
replicaBindDN: cn=srvA,o=moppssc,c=fr
replicaCredentials: pwA
replicaBindDN: cn=srvBC,o=moppssc,c=fr
replicaCredentials: pwBC
peerServerDN: cn=srvA,o=moppssc,c=fr
peerServerPW: pwA
replicaBindDN: cn=srvB,o=moppssc,c=fr
replicaCredentials: pwB
replicaBindDN: cn=srvBC,o=moppssc,c=fr
replicaCredentials: pwBC

Figure 5-47 Replication configuration example

New messages have been added to indicate possible problems between replicas. For example, if an entry is deleted on one peer but another peer cannot find the entry when it attempts to replicate the delete operation, a message is issued to indicate that there may be a problem.

Synchronizing the initial content of peer-to-peer replicas can be a problem if they are accepting update requests during the priming process. To prevent this from happening, the peer LDAP servers can be switched to maintenance mode by an MVS MODIFY command. This mode only allows operations (including searches) to be done when bound with the peerServerDN or the masterServerDN DN value. Once the contents of replica servers have been initialized, the MODIFY command can be used to switch to normal operating mode again where all requests are processed.

Note: One implication of being in maintenance mode is that the LDAP server is not accepting anymore LDAP operations by the LDAP administrator.

The command to enter maintenance mode is:

```
f ldapsrv,appl=maintmode=on
```

Which is answered with the messages shown in Figure 5-48.

```
GLD0264I Slapd Maintenance mode is starting.
GLD0263I Slapd is ready for requests (Maintenance mode).
```

Figure 5-48 Entering maintenance mode

Now that we are in maintenance mode, we can try to run an LDAP command issued by the LDAP administrator:

```
ldapsearch -h 10.11.12.13-p 389 -D cn=admin -w secret -b "c=fr" "objectclass=*”
```

The response to this command is shown in Figure 5-49.

The following command runs, as though it is issued by the replication DN and succeeds:

```
ldapsearch -h 10.11.12.13-p 389 -D cn=peerServer,c=fr -w peerServer -b "c=fr"
"objectclass=*"
```

```
ldap_search: Directory server is unwilling to perform the operation
ldap_search: additional info: R007052 Error: server running in maintenance mode,
 operations restricted to peerServerDN or masterServerDN.
```

Figure 5-49 Request from the LDAP administrator while in maintenance mode

5.4 Miscellaneous improvements since z/OS 1.4

In this section we discuss more improvements since z/OS 1.4.

5.4.1 Incorrect bind DN

For security reason, to avoid a discovery of a DN by repeated searches, an invalid bind DN condition returns the code LDAP_INVALID_CREDENTIALS, instead of LDAP_NO_SUCH_OBJECT.

5.4.2 RDBM and JNDI removal

As of z/OS 1.4, the RDBM backend is no longer part of the z/OS LDAP server. The RDBM backend was the initial implementation of the LDAP data store on DB2. With OS/390 R10, LDAP introduced the TDBM backend, which optimizes the storing of data in DB2 for improvement of both performance and DB2 tables management. The TDBM backend provides all of the functionality of the RDBM backend in a highly scalable backend data store. There is a chapter in *z/OS Integrated Security Services LDAP Server Administration and Use*, SC24-5923, on migrating from RDBM to TDBM backend.

The IBM JNDI implementation is discontinued at z/OS 1.4. clients have to move to the IBM distribution of the SUN JNDI in the JDK, as it has become the most commonly used version of JNDI. More information about the Sun JNDI interface can be found at the Sun Web site:

http://www.sun.com

5.4.3 SDBM backend

In this section we briefly comment on some of the SDBM backend enhancements since z/OS 1.4.

Support of the EIM segment in the RACF user profile (z/OS 1.4)

The SDBM schema has been updated with a new object class, racfEIMSegment, and a new attribute, racfLDAPProf, which correspond, respectively, with the EIM segment and the LDAPPROF field in the RACF USER profile (refer to “LDAP and domain bind information” on page 175 for an explanation of their meaning and use). They can be the subject of add, modify, delete, or display value via LDAP commands.

Support of SHARED and AUTOUID/AUTOGID specification in OMVS segments (z/OS 1.4)

New options for controlling the use of shared UNIX identities and automatic ID assignment have been introduced in RACF at z/OS 1.4 level (refer to 2.3, “UNIX identity management” on page 16, for a description of these functions). The RACF USER OMVS segment support in the SDBM has been updated to take into account two new attributes:

- racfOmvsUidKeyword, optional in the racfUserOmvsSegment object class
- racfOmvsGroupIdKeyword, optional in the racfGroupOmvsSegment object class

They may have a value of SHARED, AUTOUID, or AUTOGID.

The attributes may be used only when adding or modifying an entry. Additionally, this enhancement requires the RACF database to be at AIM Stage 2 to operate, or the request will fail.

Enhancement to support RACF search (z/OS 1.4)

The filters initially made available searching a USER or GROUP profile by UID or GID are:

- racfomvsuid=*nnn*
- racfomvsgroupid=*nnn*

Where *nnn* is the searched for UID or GID. However, the search completes at the first found occurrence of nnn.

New options are given to these filters at z/OS 1.4. They can be expressed as:

- racfomvsuid;allOMVSids=nnn
- racfomvsgroupid;allOMVSids=nnn

The search result this time is a full list of all the USER or GROUP DNs that match the criteria. For example, the **ldapsearch** command, issued with:

```
-b "sysplex=mv08" "racfomvsuid;allOMVSids=045"
```

That is, here, a search starting at the root base with a subtree scope. This search yields in our SDBM directory:

```
racfid=YURI,profiletype=USER,sysplex=MV08  racfid=YURY,profiletype=USER,sysplex=MV08
racfid=YURY1,profiletype=USER,sysplex=MV08
```

This enhancement requires the RACF database at AIM Stage 2 or the request will fail.

5.4.4 TDBM backend DB2 restart and recovery

Prior to z/OS 1.6 LDAP, the LDAP server did not react when the DB2 subsystem terminated, which resulted in the LDAP client failing and the LDAP administrator unaware of the problem. Now when using the TDBM or GDBM backend, the LDAP server has an improved availability by monitoring DB2 and detecting when it shuts down. We can now use the configuration file to configure how the LDAP server will respond when DB2 terminates with the db2terminate directive.

We have one of two options that we can code:

- Terminate - The LDAP server will shut down.
- Restore - The LDAP server will disconnect from DB2 but remain running to allow access to non-DB2 backends (for example, SDBM). When DB2 is once again active, the LDAP server will re-connect to DB2. There will be no access allowed to DB2-based backends (TDBM and GDBM) during the time when DB2 is down.

The default is restore. Note that If using a sysplex distributor, this configuration option should be set to terminate. This will allow client requests to be routed to other LDAP servers in the sysplex who can connect to their databases.

Example of terminate option

We edited the configuration file with the terminate option and re-cycled the LDAP server to pick up our change. We then stopped DB2. On the system console we can see straight after the DB2 stop command, that the LDAP server is terminating. This is shown in Figure 5-50. Figure 5-51 is a view of the LDAP server started task sysout.

```
-DBS1 STOP DB2
BPXM023I (GLDSRV) 592
GLD0004I Terminating slapd.

BPXM023I (GLDSRV) 593
GLD0236I IP address: INADDR_ANY, port 389 has been stopped.

DSNY002I  -DBS1 SUBSYSTEM STOPPING
$HASP395 GLDSRV   ENDED
```

Figure 5-50 Messages in the system log after DB2 has terminated

```
GLD0251E DB2 termination detected, server is stopping
GLD0004I Terminating slapd.
GLD0236I IP address: INADDR_ANY, port 389 has been stopped.
```

Figure 5-51 Messages in the LDAP started task sysout

Example of restore option

We edited the configuration file to set up db2terminate with the *restore* value and re-cycled the LDAP server to pick up our change. We then stopped DB2. This time there is no indication in the system log that something went wrong, but the information shown in Figure 5-52 on page 147 appears in the LDAP started task sysout. An attempt to initiate a data retrieval operation at the LDAP server, as can be done from an LDAP browser, results in receiving the message shown in Figure 5-53 on page 147.

```
GLD0252E DB2 termination detected, database access unavailable.
```

Figure 5-52 DB2 termination indication in the LDAP server sysout

```
06:26:36 PM: List failed
Root error: [LDAP: error code 1 - R004165 Error creating request structure - DB2
Unavailable
. (tdbm_search.c|1.74.1.53|599)]
```

Figure 5-53 Error seen through a browser

When DB2 is restarted, the LDAP server picks this up through the monitoring of DB2. The LDAP server started task sysout is updated with the message:

```
GLD0253I DB2 restart detected, database access available.
```

The db2terminate configuration is ignored if no DB2 backend is configured.

5.5 LDAP Client - Miscellaneous improvements

The LDAP client has been rewritten to provide more complete support for various RFCs and to provide 64-bit capability. This has resulted in improved performance, and serviceability, enhanced functionality, and positions the client for future enhancements.

5.5.1 SOCKS V5 support

The LDAP client was already socksified for SOCKS V4. It now supports SOCKS V5 beginning with z/OS 1.6. This supports user authentication and IPv6 addressing. The implementation is based upon the following RFCs:

- RFC 1928 - SOCKS Protocol V5
- RFC 1929 - Username/password authentication for SOCKS V5
- RFC 2373 - IPv6 addressing
- RFC 2732 - Literal IPv6 addresses in URLs

Refer to *z/OS Integrated Security Services LDAP Client Programming*, SC24-5924.

5.5.2 Enhanced security functions

In this section we discuss enhanced security functions.

Start and stop TLS

The client API has been updated, at z/OS 1.6, to support the start and stop of the transport layer security. This is provided with the ldap_start_tls_s_np and ldap_stop_tls_np functions. This allows a connection to a server to handle SSL and non-SSL. One use for this would be to do authentication to a server over a non-SSL connection, using, for instance, the ldap_simple_bind or ldap_sasl_bind. Later if data confidentiality or integrity is wanted, ldap_start_tls_np can be issued for the same ldap connection handle. This does not change the authentication for the connection.

SASL security layer maximum and minimum security levels

SASL security layer support integrity and confidentiality has changed, at the z/OS 1.6, with the addition of maximum and minimum integrity and confidentiality levels can be set via the ldap set option APIs. The options that can be set are:

- LDAP_OPT_MAX_SASL_LEVEL, which is the highest level that will be negotiated during the bind
- and LDAP_OPT_MIN_SASL_LEVEL, which is the minimum level that will be negotiated during a bind

Levels of protection that can be asked for are no integrity or confidentiality protections, integrity protection only, or integrity and confidentiality protection.

These options are currently only used if the mechanism of GSSAPI or DIGEST-MD5 is specified on the ldap_sasl_bind or the ldap_sasl_bind_s APIs and the target LDAP server supports integrity and confidentiality.

For the EXTERNAL, simple, and CRAM-MD5 bind mechanism, SSL/TLS provides integrity and confidentiality services.

5.5.3 Extended processing for distinguished names

The following additional LDAP APIs are provided for processing distinguished names (DNs):

- To break the distinguished name into one or more relative distinguished names in different formats:
  - ldap_dn2ufn, ldap_dn2ufn_np
  - ldap_explode_dn, ldap_explode_dn_np
- To break a relative distinguished name into one or more attributes:
  - ldap_explode_rdn
- To free the storage allocated for an LDAP DN description:
  - ldap_free_dndesc_np

5.5.4 Extended rebind processing

Extended rebind processing is specified via the ldap_set_option, ldap_set_option_np API by specifying the LDAP_OPT_EXT_REBIND_FN option and providing the name of the ext_rebind_proc() routine the LDAP client runtime needs to invoke with its needs to authenticate a connection with another LDAP server followed by a referral from an initial LDAP server. When the LDAP_AUTH_SASL authentication method is specified by the ext_rebind_proc() routine, any of the supported SASL mechanisms can be used.

RACF Password Enveloping and z/OS LDAP Change Log

In this chapter we introduce these two new functions, available at z/OS 1.6 and rolled back to z/OS 1.3 in the form of an Small Programming Enhancement (SPE) via RACF APARs OA03853 and OA03854, and LDAP APAR OA03857. These enhancements allow RACF to signal the LDAP GDBM backend (see "Change Log, as used by the TDBM backend" on page 110) of changes in a RACF USER profile—this is the *LDAP Event Notification*; and if the change affects the USER password, the new password can be extracted encrypted from the RACF database via the LDAP protocol—this is *Password Enveloping*. These operations are tightly controlled by RACF profiles that we also describe in this chapter.

Note: If you are not already at z/OS 1.6, the z/OS APAR OA03853 points to a document ("z/OS RACF Password Enveloping and LDAP event notification enhancements support") that describes the new function; the document may be downloaded from the "what's new" section of the RACF Web site at:

http://www-1.ibm.com/servers/eserver/zseries/zos/racf

Beginning with z/OS 1.6 the contents of this document are integrated inside the proper books of the RACF bibliography. It might be helpful to our readers to have a copy of the document, as we reference it through this chapter.

We intend here to demonstrate the complete setup of the new function, and what information can be collected, via the LDAP protocol, from a Change Log.

6.1 The overall view

This section provides an overall view of these two new functions, in the context of their exploitation by a device such as the IBM Directory Integrator.

Refer to Figure 6-1 for the different steps of the process:

1. A user modifies her password. The new password is processed as usual, that is, encrypted one way and stored in the USER profile. However, for users selected by the RACF administrator, the new password is also set apart encrypted with the RACF RSA public key (this function involves an RS key pair that belongs to RACF).
2. RACF also notifies the LDAP server of a change to the USER profile. The GDBM backend creates a Change Log entry in its directory, which indicates that a change in a RACF USER profile occurred.
3. It is expected in this scenario that a device like the IBM Directory Integrator is continuously searching for a change (that is, a new log entry) in the GDBM directory (see the capability of supporting LDAP persistent search at z/OS 1.6 in 5.3.5, "LDAP persistent search" on page 140).
4. Having detected a change, the IBM Directory Integrator comes back with an LDAP search to the SDBM (that is, RACF) backend to get the new value of the racfPasswordEnvelope attribute.
5. RACF proceeds with a decryption of the password stored aside, using its private key and proceeds with a new encryption of the password, based on the requestor's public key, in a format called a *PKCS#7 envelop*. Note that the requestor's public key must have been previously made available to RACF as the requestor digital certificate connected to a keyring owned by RACF.
6. The IBM Directory Integrator gets the encrypted "envelop", proceeds with its decryption, and now has the user new password that it can propagate to other directories for synchronization purposes.

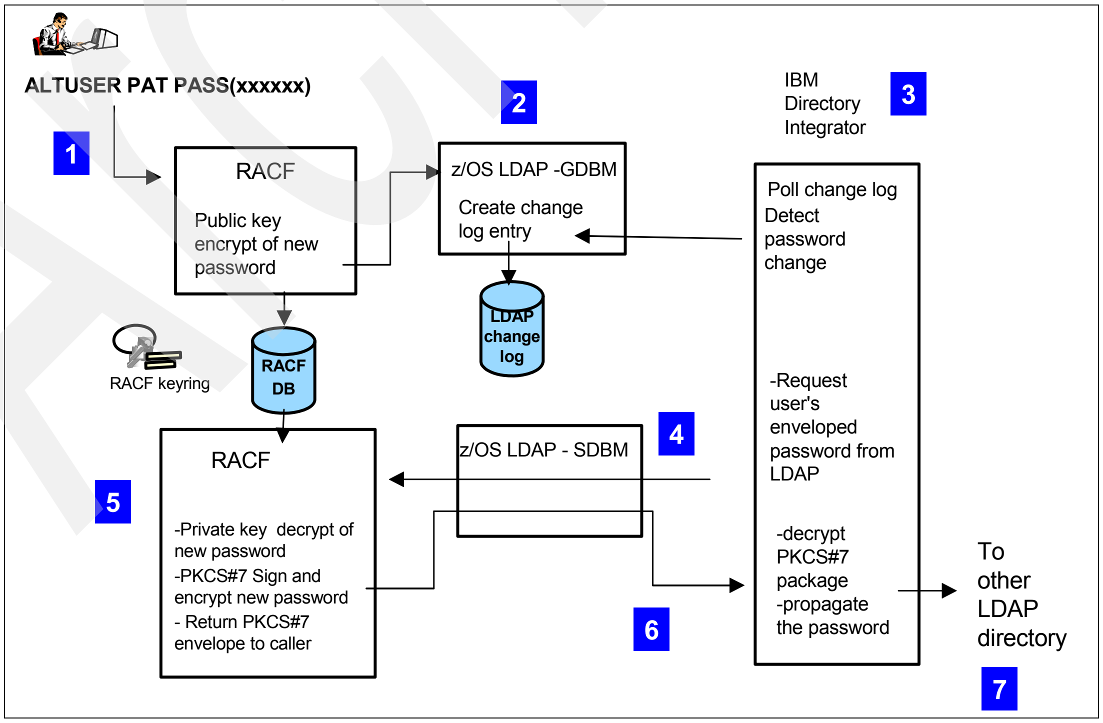

ALTUSER PAT PASS(xxxxxx)
1
RACF
Public key encrypt of new password
2
z/OS LDAP -GDBM
Create change log entry
LDAP change log
IBM Directory Integrator
3
Poll change log Detect password change
RACF keyring
RACF DB
-Request user's enveloped password from LDAP
4
z/OS LDAP - SDBM
5
RACF
-Private key decrypt of new password
-PKCS#7 Sign and encrypt new password
- Return PKCS#7 envelope to caller
6
-decrypt PKCS#7 package
-propagate the password
To other LDAP directory
7

Figure 6-1 RACF Password Enveloping and Event Notification - Overall view

6.2 Enablement of Password Enveloping in RACF

In this section we discuss the enablement of Password Enveloping in RACF.

6.2.1 What this new RACF function does

A user password can be extracted in RACF as a PCKS#7 *envelope*. A PKCS#7 envelope consists of encrypting and encapsulating data, using both public key and symmetric algorithms, so that only the intended recipient can retrieve the clear value of the data.

It is required that the RACF address space owns a RACF keyring with:

- An RSA key pair belonging to the RACF address space itself
- RSA certificates for applications that will be allowed to extract the enveloped password

The process is started when a selected RACF user changes his password. RACF, along with storing the user password with a one-way encryption as usual, also proceeds with a reversible encryption of the new password, using the RACF address space public key, and stores it in the user's profile in the new PWDENV field in the BASE segment. The encrypted password takes approximately 280 bytes in a user profile, and is not displayed by the LISTUSER command.

An application that needs to retrieve this password uses the R_admin (IRRSEQ00) interface to get an encrypted password envelope (the z/OS LDAP SDBM backend is such an application). To create the envelope, RACF decrypts the password to be enveloped using its private key, and proceeds with the creation of the PKCS#7 envelope, which roughly consists of:

1. Encrypting the password with a symmetric algorithm, using a dynamically generated key
2. Encrypting the dynamically generated symmetric key with the recipient's public key, which RACF must find in its keyring

Note: Information about the PKCS#7 standard can be found at:

`http://www.rsasecurity.com/rsalabs`

The envelope is returned to the application, which then may initiate its own processing to retrieve the clear password by running in reverse order the two phases above, using its private key for the asymmetric piece of it.

RACF permissions

Obviously, RACF does not apply this processing in each and every case of changing passwords. Firstly, it does this for only those user IDs that are permitted to create password envelopes via the access list in the PASSWORD.ENVELOPE profile, defined in the new RACF class RACFEVNT. Secondly, a new password is *not* enveloped in the following situations:

- This is an initial ADDUSER password.
- When the new password is the same as the current password.
- When the ALTUSER or PASSWORD command is used to change the password, and the new password is equal the user's default group name.
- When an application uses RACROUTE or ICHEINTY (as opposed to a RACF command) to set the password, and the password contains characters that would not be accepted by the RACF commands. RACF commands only accept the characters 'A'–'Z', '0'–'9', and the variant characters X'5B' (typically '$'), X'7B' (typically '#'), and X'7C' (typically '@').

- When an application uses RACROUTE or ICHEINTY to set the password and specifies ENCRYPT=NO.

Thirdly, applications, authorized to retrieve enveloped passwords, have to provide their certificates for adding them to a special keyring, IRR.PWENV.KEYRING, in the DIGTRING class, associated with the RACF subsystem identity. And last, in order to retrieve the envelopes the applications must have READ access to profile IRR.RADMIN.EXTRACT.PWENV in the FACILITY class.

Here is the working example of Password Enveloping. During the testing we wrote a simple program to retrieve an envelope, and then tested the whole setup. Please note that we are not going into details of the implementation here, because the previously mentioned downloadable document provides them.

6.2.2 Our setup and test

On our test system the RACF Subsystem (RASP) is assigned user ID SUPUSER, which has been given an OMVS segment with a UID.

SUPUSER has READ permission to the IRR.DIGTCERT.LISTRING profile in the FACILITY class. This allows SUPUSER (that is, RACF) to use certificates connected to a keyring. Note that if your RACF started procedure is TRUSTED or PRIVILEGED then this permission is not required.

Also, it is important to note that, in order to create the envelope, RACF uses C/C++ APIs provided by z/OS System SSL. As a consequence, the RACF address space needs an OMVS segment with a UID. In order to minimize the operational changes, RACF will “dub” itself as a UNIX process (that is, it needs a UNIX UID) only if the Password Enveloping function is configured.

First of all, we created a Certification Authority key pair in RACF. Then we created a key pair and certificate intended to be owned by RACF. The RACF certificate being signed by the CA we just created.

Ensure that the newly created RACF certificate is TRUSTED. You can use the RACDCERT LIST ID(userid) command to prove this.

We then connect the RACF certificate to the keyring owned by the RACF address space as the default certificate, which is a mandatory requirement. Figure 6-2 on page 153 shows the RACDCERT commands we issued to achieve this. Notice that the generated private keys are securely kept in the ICSF PKDS (Public Key DataSet) since we used the “ICSF” keyword with the RACDCERT command. Note, however, that the ICSF option requires the hardware cryptographic coprocessors and the ICSF started task to be in operation in the system, the alternative being not to use this option and then the RACF private key is kept in the RACF database.

```
RACDCERT CERTAUTH GENCERT SUBJECTSDN(CN('RACFCA') -
O('ibm')  C('fr')) WITHLABEL('RACFCA') NOTAFTER(DATE(2020-12-31)) ICSF

RACDCERT ID(SUPUSER) GENCERT  -
SUBJECTSDN(CN('RACF AddrsSpace System') O('ibm') -
C('fr')) WITHLABEL('RASP') SIGNWITH(CERTAUTH LABEL('RACFCA')) -
KEYUSAGE(HANDSHAKE DATAENCRYPT DOCSIGN) -
NOTAFTER(DATE(2020-12-31)) ICSF

RACDCERT ID(SUPUSER) CONNECT(LABEL('RASP') -
RING(IRR.PWENV.KEYRING) DEFAULT  USAGE(PERSONAL))
```

Figure 6-2   Generating a certificate for Password Enveloping, using a new CA certificate

We defined the profile PASSWORD.ENVELOPE in the new RACFEVNT class. We have already mentioned that the profile tells RACF if a user is eligible for Password Enveloping when his password has been changed. The user in such case should have at least READ access to the PASSWORD.ENVELOPE profile. These are the commands below:

```
RDEF RACFEVNT PASSWORD.ENVELOPE UACC(NONE)
PE PASSWORD.ENVELOPE CL(RACFEVNT) ID(JON YURY) ACC(READ)
```

Note that the APPLDATA of the PASSWORD.ENVELOPE profile is used to specify the signing hash algorithm used to sign the PCKS#7 password envelope, and the encryption strength used when encrypting the password envelope. If the APPLDATA is not specified, the defaults are MD5 for signing hash algorithm and STRONG for the encryption strength, which corresponds to APPLDATA('MD5/STRONG'). The following values are allowed for the signing hash algorithm:

- MD5 - This is the default
- SHA1

The following values are allowed for the encryption strength:

- STRONG - This is the default - triple DES encryption (168-bit key).
- MEDIUM - Single DES encryption (56-bit key).
- WEAK - RC2 40-bit key.

It is recommended that you use the strongest encryption possible. You can use STRONG if you have installed the z/OS Cryptographic Services Security Level 3 FMID. Getting this FMID in your country of use might be restricted by local regulations.

Now we can activate and RACLIST the RACFEVNT class:

```
SETR CLASSACT(RACFEVNT)
SETR RACLIST(RACFEVNT)
```

Note: Stop and restart the RACF subsystem address space after defining the PASSWORD.ENVELOPE profile and activating the RACFEVNT class. The RACF address space can be stopped and restarted using the following commands:

```
#STOP
S RACF,SUB=MSTR
```

This assumes that the RACF subsystem prefix is #, and the RACF started task is "RACF".

Stopping and re-starting the RACF address space needs to be done, for, as we have mentioned earlier, RACF will only *dub* itself as a UNIX process if the Password Enveloping function is configured, and RACF determines this during subsystem initialization. Hence, if

Password Enveloping is configured on the fly, the RACF address space must be stopped and restarted in order for it to create the proper UNIX system services environment.

To finish the brief discussion of the underlying mechanism we have to notice that during system IPL, the RACF subsystem address space must wait for the OMVS kernel to initialize, and so the address space may not be available until later in the IPL sequence than it otherwise would have if Password Enveloping were not configured.

Returning to our test, the last step is to define profile IRR.RADMIN.EXTRACT.PWENV in the FACILITY class. The profile will be used to control which applications (that, is user IDs) are eligible to retrieve enveloped passwords. We do not define any logging options, although we strongly recommend to turning auditing on:

```
RDEF FACILITY IRR.RADMIN.EXTRACT.PWENV UACC(NONE)
```

Of course, if any of the mentioned classes is RACLISTed, then each time there is a change in the class, it needs to be refreshed.

When it comes to retrieve the password envelope using the R_Admin callable service, it is possible to do so via a program running in supervisor mode only.

However, if such a program runs under a user ID that has not provided its certificate to be connected to RACF's IRR.PWENV.KEYRING keyring, expect to receive SAFRC=4 RACFRC=4 RACFRSN=8 ("For ADMN_XTR_PWENV, a problem was encountered while retrieving the password envelope."). Additionally, RACF issues the message:

```
IRRC130I ((RACF)) SYSTEM SSL FUNCTION X'00000040' RETURNED
ERROR CODE  X'03353033' DURING OPERATION NUMBER X'00000001'
WHILE PROCESSING THE PASSWORD ENVELOPE FOR USER <user>
```

Error code X'03353033' (CMS Status code) reads as `A recipient certificate not found. At least one recipient certificate is to be provided.`

Therefore, intended recipients of password envelopes must provide their certificates to RACF. We created one such certificate for our test program using the RACDCERT command, and connected it to the keyring. Notice how the KEYUSAGE attributes are specified; it is the requirement to have them specified this way:

```
RACDCERT ID(YURY) GENCERT –
SUBJECTSDN(CN('PSWD Envelope Example') O('ibm') C('fr')) –
WITHLABEL('yurycert') SIGNWITH(CERTAUTH LABEL('RACFCA')) –
KEYUSAGE(HANDSHAKE DATAENCRYPT DOCSIGN)

RACDCERT ID(SUPUSER) CONNECT(ID(YURY) LABEL('yurycert') –
RING(IRR.PWENV.KEYRING) USAGE(PERSONAL))
```

So, now we can authorize the user ID (YURY), used by the test program, to retrieve password envelopes. We do this by placing user YURY in the access control list of the FACILITY class profile IRR.RADMIN.EXTRACT.PWENV with READ access. The class may be refreshed afterwards, depending on its RACLISTed status.

Note that we have taken a simple example where the certificate for YURY has been signed by the RACFCA certification authority, which is already connected to the IRR.PWENV.KEYRING keyring.

Note, however, that RACF does not verify the recipient's Certification Authority chain. It is up to the RACF administrator to connect the recipient's certificate to the RACF keyring and to set the certificate status to TRUST.

This setup being done, the test program successfully retrieves the envelope with the password. You can find the assembler source code of our test program in “RACF Password Envelopping sample test” on page 293.

6.3 LDAP Change Notification

Now that RACF has been set up and Password Enveloping successfully tested, we turn our attention to the second part of the new function: The notification by RACF, to the z/OS LDAP server, of a change in the USER profile.

The intent is to have the change be reflected in the GDBM backend directory (see GDBM and Change Log at “Activity logging” on page 108). The changes in a RACF USER profile that result in creating a Change Log entry in the GDBM backend are:

- Password changes, regardless of the interface used, as long as the new password has to be enveloped; see the previous section.
- Updates to a user’s revoke status, that is, changes to the FLAG4 field in the USER profile, regardless of the interface used to make the change.
- User additions made using the ADDUSER command.
- User modifications made using ALTUSER and PASSWORD commands.
- User deletions made using the DELUSER command.

LDAP notification does not cover application changes, made using the RACROUTE and ICHEINTY macro. Changes to group connection information are not currently supported. Other RACF commands that affect user profiles (RACDCERT, RACLINK) are not supported either.

To have RACF notifying the LDAP server to log a change, a new profile in the RACFEVNT class is to be defined: NOTIFY.LDAP.USER. The requirement is just to have this profile defined, that is, the profile acts as a switch to turn on or off RACF event notification.

Support for logging came in the form of LDAP APAR OA03857, which introduced a new LDAP backend: The GDBM backend. GDBM provides support for a directory of Change Log entries. Regular LDAP search mechanisms can be used to view Change Log entries. To allow RACF to request creation of Change Log entries a new extended operation was added. Finally, SDBM was extended to allow an LDAP search to retrieve password envelopes.

Important: LDAP change notification is also available to any application via the R_Proxyserv (IRRSPY00) RACF callable service. R_Proxyserv requires the requestor to be in supervisor mode.

6.3.1 Testing RACF Event Notification

In this section we wish to demonstrate how LDAP notification works. The first step is enabling the function. We define the NOTIFY.LDAP.USER profile, and refresh the RACFEVNT class:

```
RDEF RACFEVNT NOTIFY.LDAP.USER UACC(NONE)
SETR RACLIST(RACFEVNT) REFRESH
```

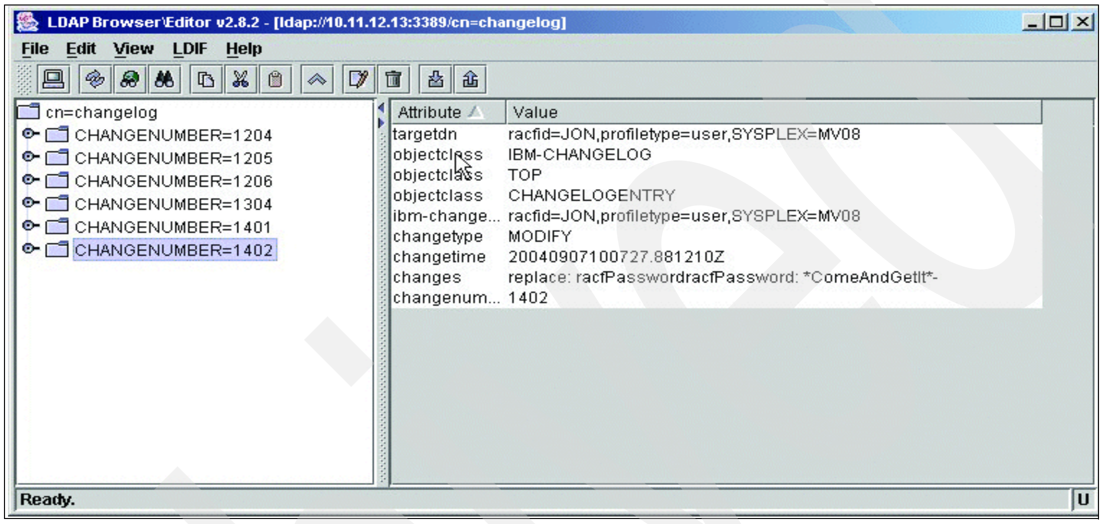
From now on, all eligible user changes will be logged by RACF in the GDBM Change Log entries.

We then edit the LDAP server configuration file to allow LDAP to accept PC calls. All we had to do was to un-comment the listen ldap://:pc directive example and ensure that we have the

GDBM backend installed and set up. We then stopped and started our LDAP server, to pick up our change. The started task sysout was showing:

```
GLD0202I Program Call communication is active.
```

We got JON to log back on to TSO and change his password. This resulted in an entry being added to the log, shown in Figure 6-3, that we can view via the LDAP browser. Note that a password change is indicated in the Change Log entry by the fixed value *ComeAndGetIt* for the racfPassword attribute.

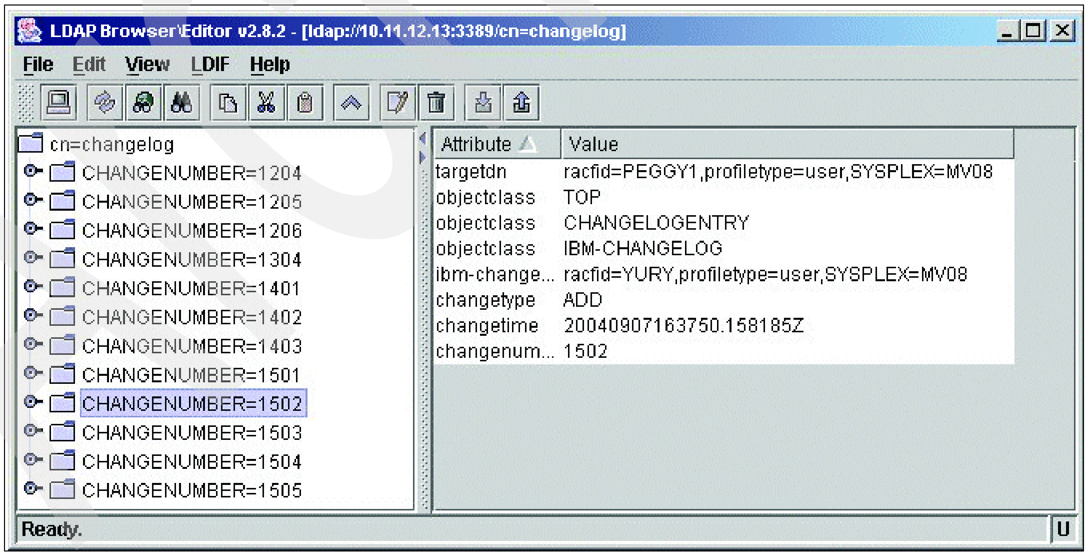

LDAP Browser\Editor v2.8.2 - [ldap://10.11.12.13:3389/cn=changelog]
File Edit View LDIF Help
Ready.

```
cn=changelog
  CHANGENUMBER=1204
  CHANGENUMBER=1205
  CHANGENUMBER=1206
  CHANGENUMBER=1304
  CHANGENUMBER=1401
  CHANGENUMBER=1402
```

Attribute	Value
targetdn	racfid=JON,profiletype=user,SYSPLEX=MV08
objectclass	IBM-CHANGELOG
objectclass	TOP
objectclass	CHANGELOGENTRY
ibm-change...	racfid=JON,profiletype=user,SYSPLEX=MV08
changetype	MODIFY
changetime	20040907100727.881210Z
changes	replace: racfPasswordracfPassword: *ComeAndGetIt*-
changenum...	1402

Figure 6-3 Change Log: Password change

We then added a new user ID, PEGGY1. A new Change Log entry is created, as shown in Figure 6-4.

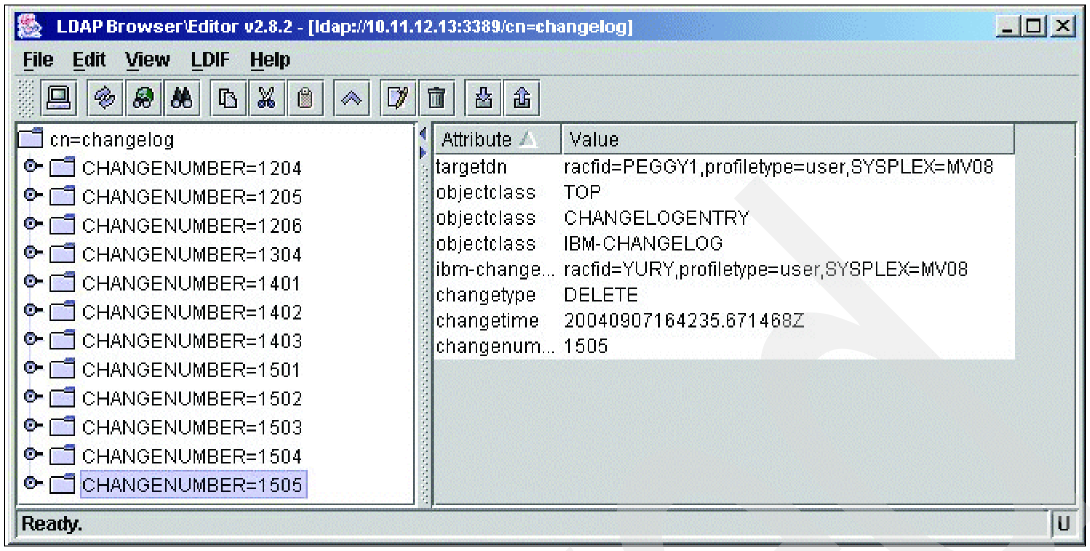

LDAP Browser\Editor v2.8.2 - [ldap://10.11.12.13:3389/cn=changelog]
File Edit View LDIF Help
Ready.

```
cn=changelog
  CHANGENUMBER=1204
  CHANGENUMBER=1205
  CHANGENUMBER=1206
  CHANGENUMBER=1304
  CHANGENUMBER=1401
  CHANGENUMBER=1402
  CHANGENUMBER=1403
  CHANGENUMBER=1501
  CHANGENUMBER=1502
  CHANGENUMBER=1503
  CHANGENUMBER=1504
  CHANGENUMBER=1505
```

Attribute	Value
targetdn	racfid=PEGGY1,profiletype=user,SYSPLEX=MV08
objectclass	TOP
objectclass	CHANGELOGENTRY
objectclass	IBM-CHANGELOG
ibm-change...	racfid=YURY,profiletype=user,SYSPLEX=MV08
changetype	ADD
changetime	20040907163750.158185Z
changenum...	1502

Figure 6-4 Change Log: Add user ID

Then we deleted the newly created user ID PEGGY1, which resulted into the log entry in Figure 6-5 on page 157.

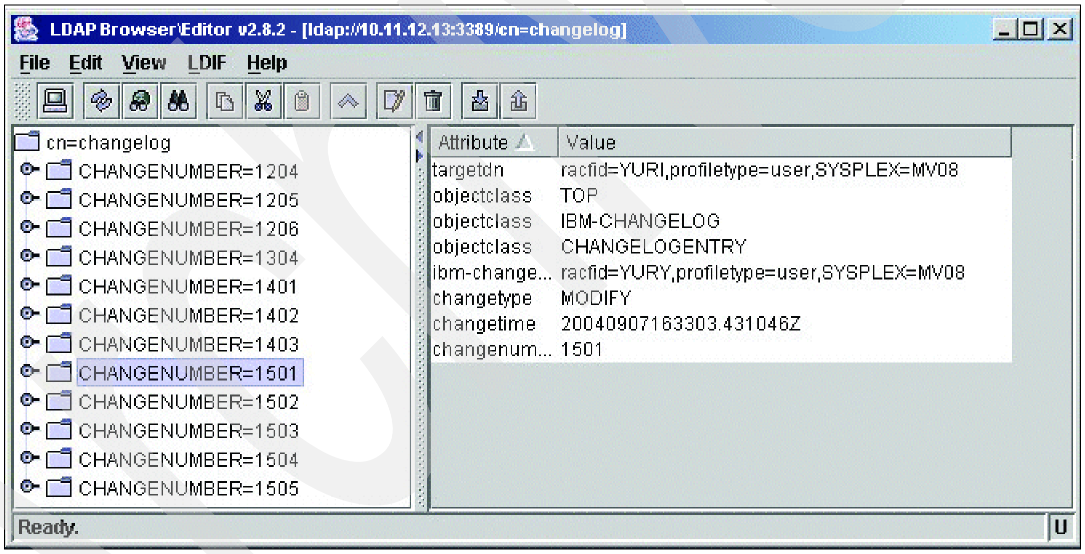

LDAP Browser\Editor v2.8.2 - [ldap://10.11.12.13:3389/cn=changelog]
File Edit View LDIF Help
Ready.

cn=changelog
CHANGENUMBER=1204
CHANGENUMBER=1205
CHANGENUMBER=1206
CHANGENUMBER=1304
CHANGENUMBER=1401
CHANGENUMBER=1402
CHANGENUMBER=1403
CHANGENUMBER=1501
CHANGENUMBER=1502
CHANGENUMBER=1503
CHANGENUMBER=1504
CHANGENUMBER=1505

Attribute	Value
targetdn	racfid=PEGGY1,profiletype=user,SYSPLEX=MV08
objectclass	TOP
objectclass	CHANGELOGENTRY
objectclass	IBM-CHANGELOG
ibm-change...	racfid=YURY,profiletype=user,SYSPLEX=MV08
changetype	DELETE
changetime	20040907164235.671468Z
changenum...	1505

Figure 6-5 Change Log: Delete user ID

The last screen shot, in Figure 6-6, is the view of a typical log entry, created by RACF for such USER changes as REVOKE, RESUME, addition of a new segment, etc.


LDAP Browser\Editor v2.8.2 - [ldap://10.11.12.13:3389/cn=changelog]
File Edit View LDIF Help
Ready.

cn=changelog
CHANGENUMBER=1204
CHANGENUMBER=1205
CHANGENUMBER=1206
CHANGENUMBER=1304
CHANGENUMBER=1401
CHANGENUMBER=1402
CHANGENUMBER=1403
CHANGENUMBER=1501
CHANGENUMBER=1502
CHANGENUMBER=1503
CHANGENUMBER=1504
CHANGENUMBER=1505

Attribute	Value
targetdn	racfid=YURI,profiletype=user,SYSPLEX=MV08
objectclass	TOP
objectclass	IBM-CHANGELOG
objectclass	CHANGELOGENTRY
ibm-change...	racfid=YURY,profiletype=user,SYSPLEX=MV08
changetype	MODIFY
changetime	20040907163303.431046Z
changenum...	1501

Figure 6-6 Change Log entry for modify

One may find these entries very sketchy in terms of the included information. Therefore, let us note that this was designed this way to provide for just a notification that a change occurred (though password changes can be differentiated). Thus, an application must maintain its own record of the state of the user if a delta is to be calculated.

6.3.2 Retrieving the enveloped password

The enveloped password can be picked up by an LDAP search request to the SDBM backend running on the same system. The envelop is the value of a new SDBM attribute, racfPasswordEnvelope, which can be retrieved from the USER profile SDBM entry. Note that racfPasswordEnvelope must be specified in LDAP search, that is, it is not returned with the entire set of entry attributes when an attribute list is not specified. It may not be specified in an LDAP add, modify, or delete operation.

This is a binary attribute; the value itself, when retrieved, is the base-64 encoded value of the envelope.

Typically the LDAP search command would be, for example:

```
ldapsearch  –D “racfid=YURY,profiletype=user,sysplex=mv08“ –w passwd –L -b
“racfid=JON,profiletype=user,sysplex=mv08”      “objectclass=*”  racfpasswordenvelope
```

The command yields as a result, in this example:

```
dn: racfid=JON,profiletype=USER,sysplex=mv08
racfpasswordenvelope: base-64_encoded_password_envelope
```

Note that YURY authenticates to the LDAP server with a RACF DN and a RACF password, that can be optionally protected by SSL/TLS. As already mentioned, a digital certificate for YURY must be connected to the IRR.PWENV.KEYRING keyring, or the envelope cannot be created for this user.

Note that the LDAP search for racfPasswordEnvelope itself does not require you to have a GDBM backend configured.

7

z/OS Enterprise Identity Mapping (EIM) in a nutshell

In this chapter, we are describing what Enterprise Identity Mapping is, and we provide you with a short demo application of EIM using the z/OS http server. This chapter contains:

- Background information about EIM, up to z/OS 1.6
- An example of EIM exploitation with:
  - A description of the application
  - A sample code
  - Installation prerequisites and system setup information

Note that detailed information about EIM for z/OS is found in the reference book *z/OS Integrated Security Services Enterprise Identity Mapping (EIM) Guide and Reference*, SA22-7875.

7.1 What Enterprise Identity Mapping is

In this section we discuss Enterprise Identity Mapping.

7.1.1 The problem that is addressed

Today's network environments are made up of a complex group of systems and applications, resulting in the need to manage multiple user registries. These user registries are intended to be used by applications to achieve user identification and authentication. The authenticated user ID is eventually used for access control as performed by the application itself, or the middleware it runs on, or by the local operating system. In today's typical heterogeneous configurations, this ends up in the situation shown in Figure 7-1. In this figure the many different platforms in the installation are represented with the local identity they have been set up with for the user John N Smith. For instance, John N Smith is known as JohnSM in the AIX® user registry, implying that John N Smith must be identified as JohnSM by applications running on his behalf on the AIX platforms. Also in this example, applications running on behalf of John N Smith on the z/OS platform must use the SAF user ID SMITH1.

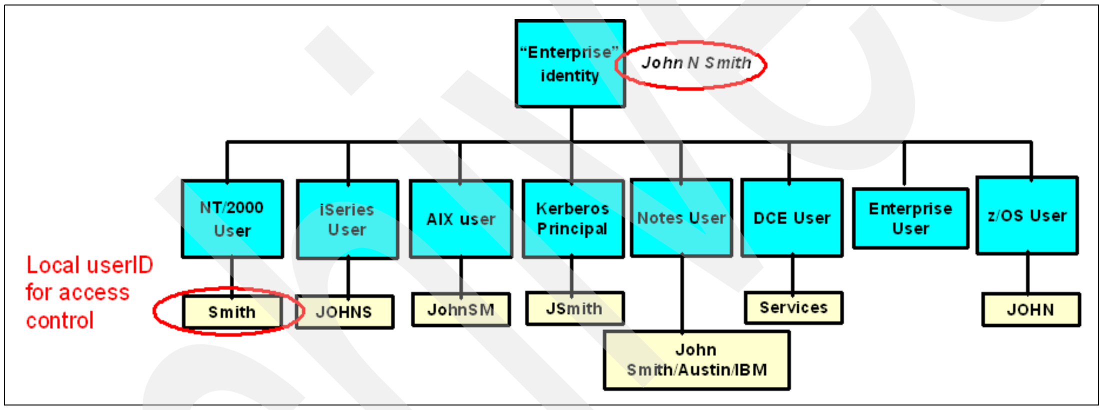


Figure 7-1 An heterogeneous configuration

Typically these multiple user registries have a design and implementation specific to each type of platform, or even application since some of them may have been implemented at the application level. They have grown over time along with the user population and the introduction of new systems and applications, yielding large administrative problems that affect users, administrators, and application developers when it comes to keeping track of a single entity and its many representations in the installation.

One must also realize that systems-specific local user IDs and registries, with their underlying mechanisms, are not going to unify in a common format across all vendors, at least for the next foreseeable many years. What seems the most natural way to proceed is for a user to authenticate to an installation using a *network identity*; it would be John N Smith in this example, and it would then be left to the involved applications to map this network identity to the locally meaningful representation (that is, user ID) of this user.

Enterprise Identity Mapping was introduced as part of the IBM Autonomic Computing Initiative and is an IBM @server infrastructure technology that allows administrators and application developers to address this problem more easily and inexpensively than previously possible. By providing centralized and protected identification facilities it also contributes to enable products and applications to run in a more secure way in today's extremely open environments, as expected for the On Demand operating infrastructure.

EIM accomplishes this by providing a central place to store mappings between user IDs that are defined in different registries in an installation. These mappings indicate:

- A relationship between user IDs, that is, user IDs that belong to the same person or entity within the enterprise.
- Or the mapping can represent a transformation of one or more user IDs to an application-specific user ID.

In the EIM terminology, the unique name given at an enterprise level for a user or an entity is the EIM Identifier. EIM actually defines associations between an EIM identifier and user IDs in registries that are part of OS platforms, applications, and middlewares.

Applications, typically servers, can then use an EIM API to find a mapping that transforms the installation level user ID initially used for authentication to a local user ID, which can in turn be used to access local resources.

Important: As the name implies, EIM is to provide identity mapping only. EIM is not to take care of the user authentication itself, which must obviously be performed ahead of identity mapping. It is left to the application deployment strategy to decide on how user authentication should be performed and how inter-application trust can be implemented. It appears today that Kerberos authentication nicely fits the authentication needs in an EIM environment.

7.1.2 The benefits of the EIM approach

The benefits are:

- The application server does not need to invent yet another user registry; it can use existing protocols for authenticating users and existing resource access managers to control the use of resources.
- The application server does not need to store these mappings in side files.
- The mappings can be used by more than one application.
- The mappings can span different platforms in the enterprise.
- Multi-tier applications can be written that will not ask the user for a platform-specific identity. They can get this identity without going into an identification and authentication process that may expose passwords.

Note: EIM is often mistaken for a user ID management solution similar to products like the Tivoli Identity Manager. An identity management solution is able to work with all aspects of a user ID, passwords, and user authorities. EIM only contains the names of user IDs and none of the other attributes. It has insufficient information for authentication or authorization.

7.1.3 The EIM implementation concepts

A simplified view of the EIM conceptual implementation is shown in Figure 7-2 on page 163. In this figure, the installation is composed of a z/OS, two OS/400®, and one AIX system. The user ID mapping is kept in an LDAP directory called an EIM Domain Controller accessible by remote EIM clients (all these terms are explained in more detail below). In this example John Smith (of whose EIM identifier is John N Smith) is running a client application on his workstation, that is eventually to send a request to a server running in SYSA. Both client application and server are using Kerberos authentication.

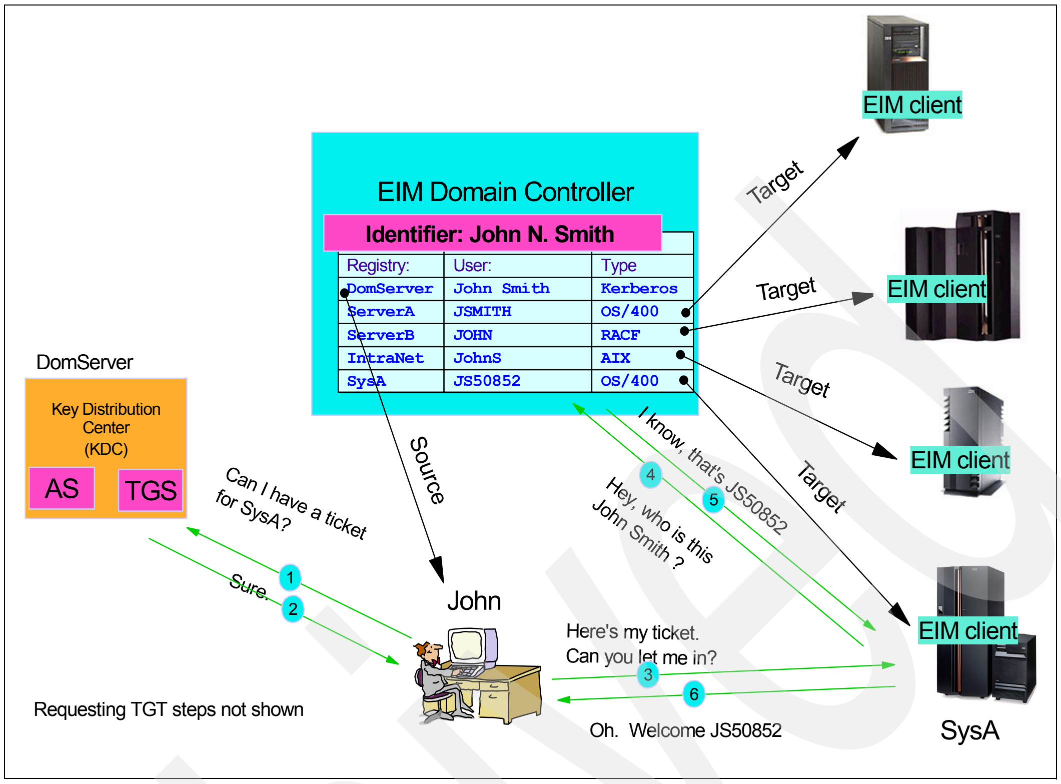
The process is:

1. John Smith has authenticated to the Kerberos authentication server (not shown in the picture) and now requests a service ticket to the Kerberos Ticket Granting Server to get access to the server on SYSA.
2. The Kerberos service ticket is delivered to the client application and contains John Smith's Kerberos principal name that we assume for this example to be John Smith@DomServer.
3. John Smith's client application requests a service on SYSA, using the Kerberos service ticket to authenticate. The Kerberos-enabled server on SYSA decrypts and parses the service ticket and extracts John Smith's principal name, John Smith, which is expected to be meaningless to the local OS/400 user registry as it has not been implemented with Kerberos semantics. Note that the successful ticket parsing process is also a proof of proper client authentication by the Kerberos authentication server.
4. The server application on SYSA then acts as an EIM client, using the local EIM API, and requests the EIM domain controller to map the principal name John Smith from source DomServer to a user ID in the target SYSA registry.
5. The EIM domain controller looks up its directory and finds the mapping information for the specified source and target identities. In this example the principal name John Smith returns JS50852 for John N Smith's SYSA local identity.

   Note: The EIM domain Controller actually proceeds by first mapping the source identity, John Smith in this example, to the EIM identifier, John N Smith, then maps the EIM identifier to the corresponding user ID in the target registry, that is, SYSA's registry.

6. The server application on SYSA then acknowledges John Smith authentication and identity mapping to the client application. It is expected that the SYSA server application is now to run on behalf of user JS50852.

In this model, it is expected that if the request had to be passed on to another tier for additional processing, SYSA would pass it with the initially authenticated identity, that is, John Smith from DomServer to the other system, which in turn would invoke the EIM Domain Controller for a local mapping.


EIM Domain Controller
Identifier: John N. Smith
DomServer
Key Distribution Center (KDC)
AS
TGS
Can I have a ticket for SysA?
Sure.
1
2
Source
John
Requesting TGT steps not shown
I know, that's JS50852
Hey, who is this John Smith ?
4
5
Here's my ticket. Can you let me in?
3
6
Oh. Welcome JS50852
Target
Target
Target
Target
EIM client
EIM client
EIM client
EIM client
SysA

Registry:	User:	Type
DomServer	John Smith	Kerberos
ServerA	JSMITH	OS/400
ServerB	JOHN	RACF
IntraNet	JohnS	AIX
SysA	JS50852	OS/400

Figure 7-2 The EIM conceptual implementation

7.1.4 EIM components

The basic components of EIM are:

The EIM domain controller It is an IBM Tivoli Directory server or IBM z/OS Integrated Security Services LDAP Server that contains one or more EIM domains. These can be:

- The OS/400 V5R2 LDAP server on iSeries™
- The z/OS V1R4 LDAP server on zSeries
- The IBM Directory Services on all platforms including AIX on pSeries®, Windows® 2000 on xSeries®, and Linux.

The information in the domain controller is created and maintained with the eimadmin utility on z/OS. As of the writing of this book, only the following IBM @servers operating systems provide the eimadmin utility or equivalent administration tool:

- •OS/400 V5R2 iSeries Navigator
- •z/OS V1R4 with SPE OW57137 and later
- •AIX 5L™ V5R3

EIM domain It contains the mappings for an enterprise. It is located by the URL for the LDAP server. The name of the domain should be descriptive of the enterprise. An example of a EIM domain's URL is `ldap://some.host/ibm-eimDomainName=My Business,c=fr`.

The EIM client This is the code that implements the EIM lookup APIs and the administrative APIs that can be used for creating, modifying, displaying, and removing information from an EIM domain.

The EIM client uses the EIM API that is provided with the eServers operating systems or that are downloadable from the Internet. As of the writing of this book, the EIM client APIs are available beginning in C/C++ or Java with these minimum systems levels:

•OS/400 V5R2
•z/OS V1R4 with SPE OW57137
•AIX R5V2
•Windows 2000, with the IBM Directory 4.1 Client
•Linux SLES8 on PPC64, Red Hat 7.3 on i386, or SLES7 on zSeries; with the IBM Directory 4.1 client or OpenLDAP v2.0.23 client

Note that the Windows and Linux clients have to be downloaded from
`http://www-1.ibm.com/servers/eserver/security/eim/availability.html`

The EIM identifier This is the name that represents the unique name of an individual or entity within the enterprise. It can be a name or number or some combination of the two that represents a person or entity in your company. The EIM identifier is the anchor point for all mappings.

The EIM registry This is the logical representation of a user registry that exists on one of the systems in the network. Examples of user registries are RACF (or equivalent), LDAP (which contains bind IDs and passwords), Lotus® Notes®, etc. An EIM registry only contains user IDs and EIM information needed for mappings with EIM identifiers. It does not contain any of the other information, such as passwords or user attributes, that normally exist in a user registry.

The EIM associations These are the relationships between a user ID and an EIM identifier. There are three kinds of associations:

• Source association - The registry user ID can be used as a source, that is, a starting point, for a lookup operation. Note that it is assumed here that the source identity has been properly authenticated by the application.

•Target Association – The registry user ID can be returned as the target value for a lookup operation. It is expected that this user ID is intended to be used for access control by the application.
The EIM domain controller also provides fields for application-specific information that can be used to refine the lookup results.

▸Administrative association - The user ID is kept in the EIM Domain Controller for administration purposes only; however, it is not recognized as a source or target user ID in a mapping lookup. And it is not returned as a result of a mapping lookup.

User IDs can be defined with both source and target and, if desired, with administrative associations.

EIM policies These are a special kind of association that can be used by an EIM lookup operation. If the EIM lookup API cannot locate a specific EIM association, the API will look for a policy. The policy acts as a default or many-to-one mapping between a source registry and a target registry. There are three kinds of policies:

►A certificate filter policy associated with an x.509 registry

►A registry policy

►A domain policy

Policies have been implemented with z/OS 1.6. See 7.5, “New EIM features at z/OS 1.6” on page 183.

EIM lookup operations They retrieve a mapping from the specified EIM domain. There are three kinds of lookups:

•eimGetTargetFromSource, which returns a user ID in the target registry when a source registry and source user ID is provided.
•eimGetTargetFromidentifier returns a user ID in the target registry when only the EIM identifier is provided.
•eimGetAssociatedIdentifier returns the EIM identifier that has an association with the given registry and user ID.

Miscellaneous additional information

More details about EIM concepts can be found at the IBM @server Information Center:

http://www-1.ibm.com/servers/eserver/security/eim/

An EIM domain controller can run on any platform where the IBM Tivoli Directory Server can be installed. To get a complete list of those platforms, go to http://www.ibm.com and search for IBM Tivoli Directory Server. The z/OS Integrated Security Services LDAP Server can also serve as an EIM domain controller.

An application that uses EIM to locate mappings needs access to the EIM client. Depending upon the platform, it is either part of the operating system or available from IBM as a download. There are two kinds of EIM clients—one is written in C/C++ and the second is written in Java. The IBM @server Information center has the latest information about supported platforms and releases.

7.2 The EIM Domain Controller

An EIM controller uses an LDAP directory server that supports the LDAP (Version 3) protocol. It must also understand the following attributes:

- ibm-entryUUID attribute
- ibmattributetypes: aclEntry, aclPropagate, aclSource, entryOwner, ownerPropagate, ownerSource
- New attribute types and object classes for EIM (provided via schema updates)

The z/OS V1R4 Security Server LDAP, with APAR OW55078, is the initial z/OS level that can host an EIM Domain Controller using the TDBM backend—that is, the Domain Controller data actually resides in DB2 tables. The SDBM backend, giving controlled access to the RACF Database, is not required. New EIM functions may require new levels of LDAP EIM schemas to be installed.

The ibm-entryuuid attribute

The uuid concept is inherited from Distributed Computing Environment (DCE) and stands for Universal Unique IDentifier. This is a unique identifier generated by the LDAP server for any entry that is created or modified and does not already have a unique identifier assigned. The unique identifier is stored in the ibm-entryuuid attribute. The ibm-entryuuid attribute is replicated to servers that support the ibm-entryuuid attribute. A utility, ldapassuuids, is provided to create the ibm-entryuuids for existing entries when migrating from previous releases.

For uniqueness, the z/OS LDAP server can be set up to use a local network adapter MAC address as a seed to generate the uuids. More details can be found in the z/OS LDAP reference book *z/OS Integrated Security Services LDAP Server Administration and Use*, SC24-5923.

7.2.1 Overview of EIM interactions

These interactions are shown in Figure 7-3 on page 167.

The EIM administrator builds and manages the directory trees representing EIM domains, using the eimadmin utility, or having a program do it by calling the EIM client administrative API. Note that it is strongly recommended to let the eimadmin utility, or the administrative API, perform this management and not to try stepping in by using direct LDAP operations such as ldapadd, ldapdelete, etc.

The EIM Domain Controller LDAP directory has been set up with the EIM schemas.

An EIM-enabled application uses the EIM client API to interrogate the EIM Domain Controller. The EIM API converts the EIM function into LDAP requests that are transparently issued by the LDAP client. Notice the local registry name, as defined by the EIM and client platform administrators and as recorded in the Domain Controller, is cached in z/OS and made available to the client API.

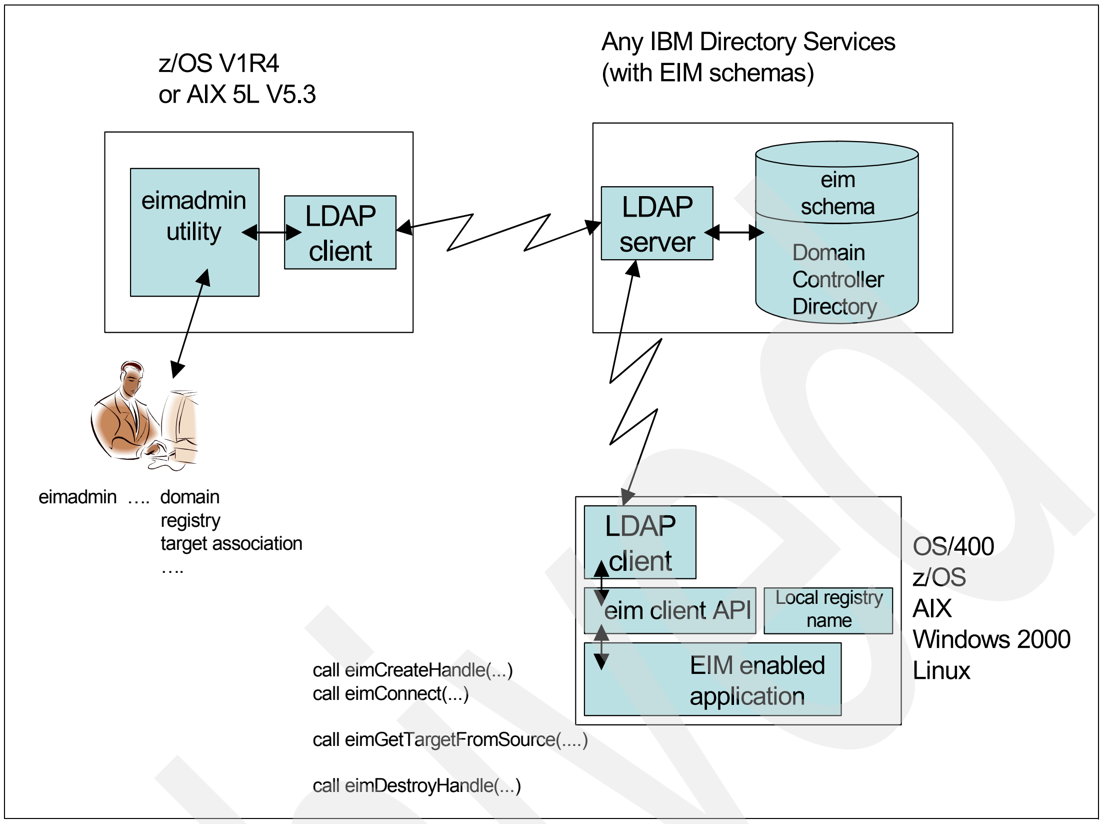


Figure 7-3 EIM infrastructure and interactions overview

The EIM APIs

There are two groups of APIs: The administration API that is used by eimadmin or a client program; and the EIM lookup API, which implements the mapping functions.

Administrative APIs

The administrative APIs are:

- Create, delete, change, and list a domain.
- Create and destroy connections with a domain.
- Add, delete, change, and list identifiers.
- Add, delete, change, and list system and application registries.
- Add, remove, and list associations.
- Change and list registry users.
- Add, remove, query, and list access to domains, identifiers, registries, and users.

Runtime lookup APIs

The runtime lookup APIs are:

- Create and destroy connections with a domain.
- eimGetTargetFromSource.
- eimGetTargetFromIdentifier.
- eimGetAssociatedIdentifiers.

A list of the EIM APIs, as of the writing of this book, is given in Appendix A, “EIM API demo sample code” on page 263.

7.2.2 Content of the EIM Domain Controller

The following objects are represented by entries in the Domain Controller:

- Domain
- Identifiers
- Registries
- Associations
- Policies

The directory tree

Figure 7-4 shows an example of a directory tree for the domain MopBooks. The subtrees to the EIM domain entry contain:

- EIM identifiers - Remember that a source user ID is pointing to an EIM identifier, and therefore all existing source associations are represented at the EIM identifier level.
- registries - These are the representations of the EIM identifier local to a system or application. The lookup target associations are represented at the registry level.
- groups - These are EIM access control groups, which are further described in "Access control to EIM objects" on page 169.

Note also the heavy use of the uuids to establish cross references between entries.

Note: Again, these directory entries are built and maintained via the eimadmin utility or the administration API. Do not try to manage directly using your own LDAP operations.

Note that the eimadmin facility accepts inputs from file. Users may consider building this file from a RACF DBUNLOAD for mass creation of user IDs mappings in an EIM domain controller.

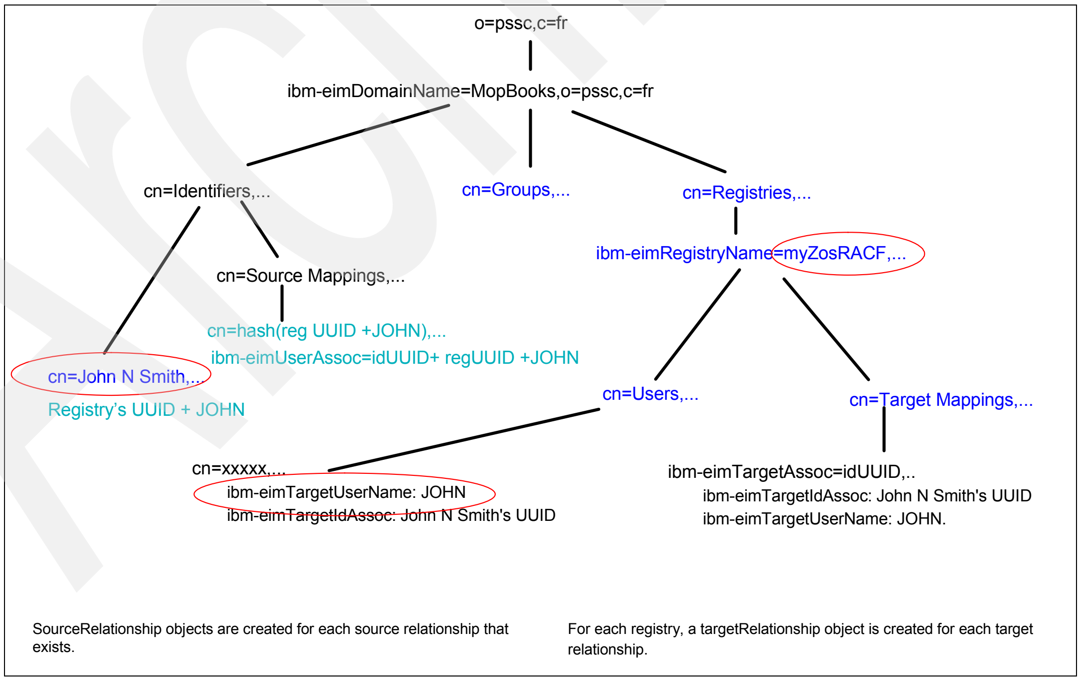

o=pssc,c=fr
ibm-eimDomainName=MopBooks,o=pssc,c=fr
cn=Identifiers,...
cn=Groups,...
cn=Registries,...
ibm-eimRegistryName=myZosRACF,...
cn=Source Mappings,...
cn=hash(reg UUID +JOHN),...
ibm-eimUserAssoc=idUUID+ regUUID +JOHN
cn=John N Smith,...
Registry's UUID + JOHN
cn=Users,...
cn=Target Mappings,...
cn=xxxxx,...
ibm-eimTargetUserName: JOHN
ibm-eimTargetIdAssoc: John N Smith's UUID
ibm-eimTargetAssoc=idUUID,..
ibm-eimTargetIdAssoc: John N Smith's UUID
ibm-eimTargetUserName: JOHN.
SourceRelationship objects are created for each source relationship that exists.
For each registry, a targetRelationship object is created for each target relationship.

Figure 7-4 An example of EIM Directory Tree

7.2.3 Access controls to the EIM Domain Controller and its contents

The access to the EIM Domain Controller and its contents is based on the accessing user's distinguished name and its proper authentication.

User authentication on z/OS LDAP

A user accessing an EIM Domain Controller running on z/OS can authenticate with:

- The LDAP simple bind - The bind distinguished name and the bind password flow over the TCP/IP session, in clear text or protected by SSL encryption.
- Alternatively, the simple bind can use a password protected by the CRAM-MD5 hashing.
- A Kerberos service ticket, with the Kerberos principal name of the client. The Kerberos protocol itself ensures proper authentication of the client.
- A client digital certificate, over an SSL connection, that contains the client distinguished name. The SSL protocol itself ensures proper authentication of the client.

Note: Much information, such as the LDAP bind and authentication data and the domain to be used, can be entered, for Security and Information Management reasons, in RACF profiles used when connecting to the EIM Domain Controller. This is explained in 7.3.1, "Using RACF profiles to keep EIM default parameters" on page 174.

Access control to EIM objects

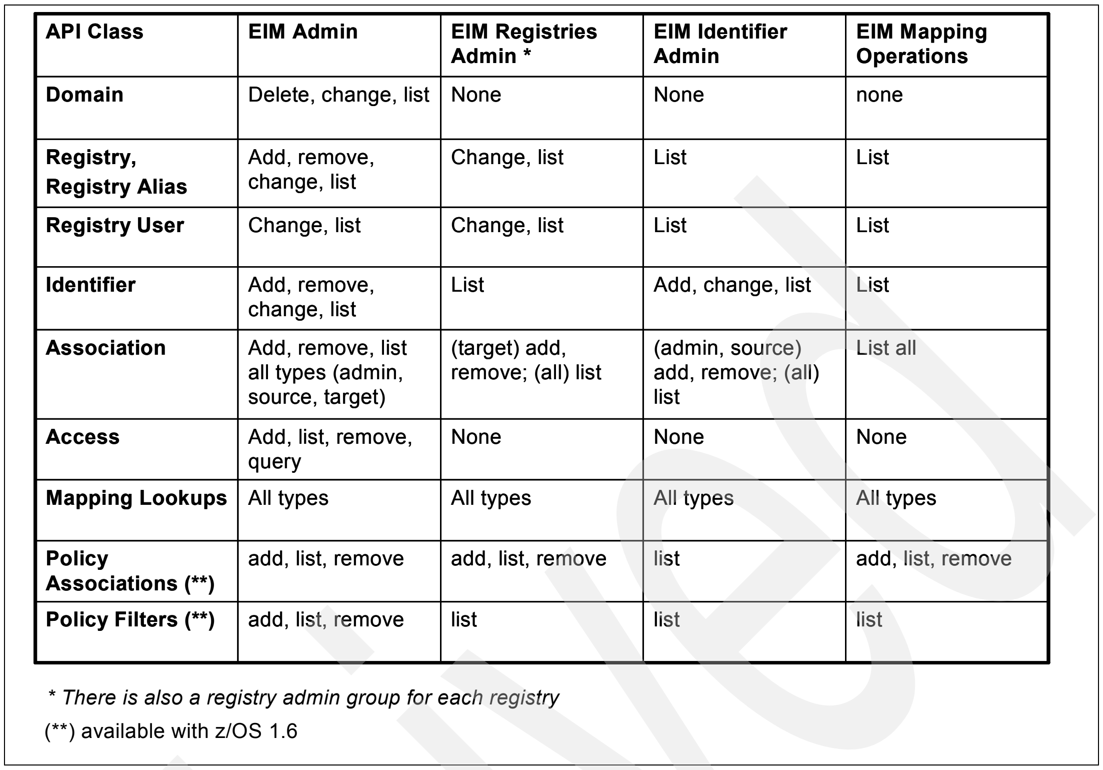
The LDAP objects in the EIM Domain Controller are subject to access controls via LDAP ACLs that grant access to the authenticated user according to the EIM access control group the user is a member of. The pre-defined access control groups are established at the EIM Domain level and are, as of the writing of this book:

- EIM Administrator - This access control group allows the user to manage all of the EIM data within this EIM domain.
- EIM Registries administrator or registry x administrator - This access control group allows the user to manage all EIM registries, or only one specific registry, definitions, and target associations.
- EIM Identifier administrator - This access control group allows the user to add and change EIM identifiers and manage source and administrative associations.
- EIM Mapping Operations - This access control group allows the user to conduct EIM lookup operations. A user with this access control can perform the following functions:
  - Perform EIM lookup operations.
  - Retrieve associations, EIM identifiers, and EIM registry definitions.

The LDAP administrator has full access to any object in the directory, and only an LDAP administrator can create a new domain.

Users are assigned to access control groups by the eimAddAccess API call, or by the eimadmin utility with the -C parameter.

A user invoking the EIM API is authorized to specific functions, as shown in Figure 7-5 on page 170.



API Class	EIM Admin	EIM Registries Admin *	EIM Identifier Admin	EIM Mapping Operations
Domain	Delete, change, list	None	None	none
Registry, Registry Alias	Add, remove, change, list	Change, list	List	List
Registry User	Change, list	Change, list	List	List
Identifier	Add, remove, change, list	List	Add, change, list	List
Association	Add, remove, list all types (admin, source, target)	(target) add, remove; (all) list	(admin, source) add, remove; (all) list	List all
Access	Add, list, remove, query	None	None	None
Mapping Lookups	All types	All types	All types	All types
Policy Associations (**)	add, list, remove	add, list, remove	list	add, list, remove
Policy Filters (**)	add, list, remove	list	list	list

** There is also a registry admin group for each registry*

(**) available with z/OS 1.6

Figure 7-5 EIM access controls

7.2.4 Setting up the LDAP directory to act as an EIM Domain controller

In this section we discuss setting up the LDAP directory to act as an EIM Domain controller.

LDAP configurations

The EIM Domain Controller can take advantage of the usual LDAP configurations, aiming at enhancing availability or simplifying the name space management. These are:

- LDAP server with replicas - The replica is another usable instance of the master directory. The master-replica configuration can be implemented at z/OS; beginning with z/OS 1.6 a peer-to-peer replication configuration can also be used.
- LDAP directory with referrals - A referral is a pointer in a directory towards another directory, so that the naming space can split across several LDAP servers. LDAP V3 referrals can be used with the z/OS LDAP server.
- To achieve very high availability, z/OS also allows you to configure a single directory to be served by multiple LDAP servers located in members of a sysplex.

Note: The choice of LDAP servers to host an EIM domain controller is dictated by the availability of the following required attributes:

- ibm-entryUUID
- aclEntry
- aclPropagate
- aclSource
- entryOwner
- entryPropagate
- entrySource

EIM schemas

In z/OS the EIM LDAP object classes and attributes definitions are delivered in the schema.IBM.ldif file in /usr/lpp/ldap/etc/. They are, as of the writing of this book:

- Objectclasses
  - ibm-eimDomain
  - ibm-eimIdentifier
  - ibm-eimRegistry
  - ibm-eimSystemRegistry
  - ibm-eimApplicationRegistry
  - ibm-eimRegistryUser
  - ibm-eimSourceRelationship
  - ibm-eimTargetRelationship
  - ibm-eimDefaultPolicy
  - ibm-eimDomainName
  - ibm-eimFilterPolicy
  - ibm-eimPolicyListAux
- Attributes
  - ibm-eimDomainName
  - ibm-eimAdditionalInformation
  - ibm-eimAdminUserAssoc
  - ibm-eimDomainVersion
  - ibm-eimRegistryAliases
  - ibm-eimRegistryEntryName
  - ibm-eimRegistryName
  - ibm-eimRegistryType
  - ibm-eimSourceUserAssoc
  - ibm-eimTargetIdAssoc
  - ibm-eimTargetUserName
  - ibm-eimUserAssoc
  - ibm-eimFilterType
  - ibm-eimFilterValue
  - ibm-eimPolicyStatus

EIM administration tools

The EIM Domain Controller administration can be performed by programs using the administration API, or can be done from the z/OS UNIX console with the eimadmin line command. An example of the eimadmin utility is shown in Figure 7-6, and a synthetic view of keywords and parameters is given in Figure 7-7 on page 172.

```
PATKAP: /u/patkap>eimadmin -lD -d "ibm-eimdomainname=MopBooks,o=moppssc,c=fr" -h
            ldap://10.11.12.13:389 -b cn=ldapadmin -w secret
                   domain name: MopBooks
                     domain DN: ibm-eimdomainname=MopBooks,o=moppssc,c=fr
                      policies: DISABLED
```

Figure 7-6 Example of eimadmin use

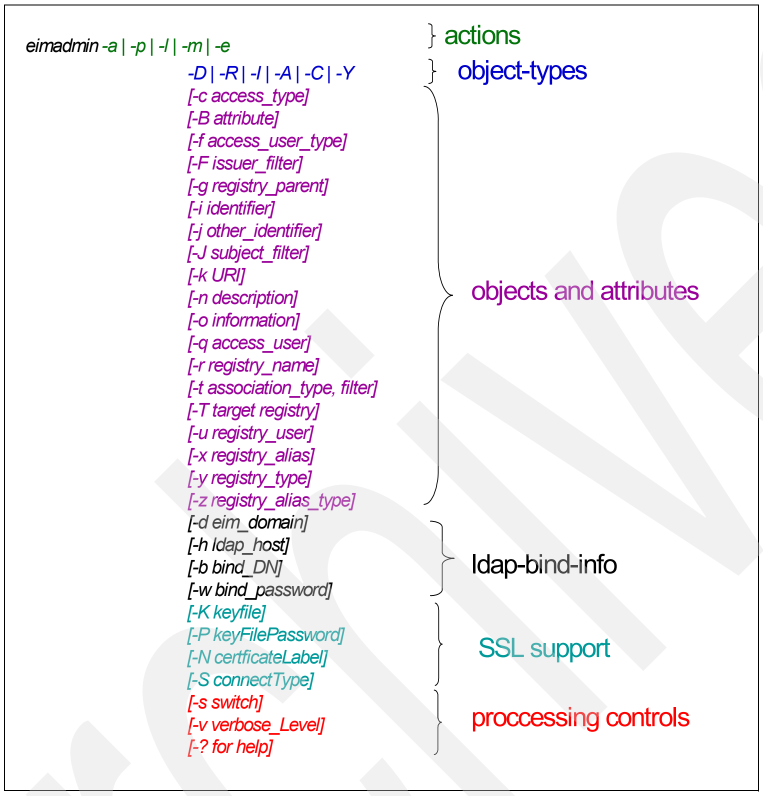
The actions that can be invoked using eimadmin are:

- Add an object.
- Delete an object.
- List objects (for example, list directories, list registries, and so forth).
- Modify attributes associated with objects.

- Erase attributes.

And objects that can be administered are:

- Domains
- Registries
- Identifiers
- Associations
- Access authorities
- Policies



```
eimadmin -a | -p | -l | -m | -e
                    -D | -R | -I | -A | -C | -Y
                    [-c access_type]
                    [-B attribute]
                    [-f access_user_type]
                    [-F issuer_filter]
                    [-g registry_parent]
                    [-i identifier]
                    [-j other_identifier]
                    [-J subject_filter]
                    [-k URI]
                    [-n description]
                    [-o information]
                    [-q access_user]
                    [-r registry_name]
                    [-t association_type, filter]
                    [-T target registry]
                    [-u registry_user]
                    [-x registry_alias]
                    [-y registry_type]
                    [-z registry_alias_type]
                    [-d eim_domain]
                    [-h ldap_host]
                    [-b bind_DN]
                    [-w bind_password]
                    [-K keyfile]
                    [-P keyFilePassword]
                    [-N certficateLabel]
                    [-S connectType]
                    [-s switch]
                    [-v verbose_Level]
                    [-? for help]
```

Figure 7-7 eimadmin utility and parameters

Note that the eimadmin utility can use an input file containing several actions that will all be performed on the single invocation of the utility.

An interesting attribute is -y, which is the registry type. As of the writing of this book, the -y attribute can have the following pre-defined values:

- RACF
- OS400
- KERBEROS (for case ignore)
- KERBEROSX (for case exact)
- AIX
- NDS
- LDAP
- PD (Policy Director)
- WIN2K
- X509
- LINUX
- DOMINOS
- DOMINOL

Refer to the chapter “EIM registry definition” in *z/OS Integrated Security Services Enterprise Identity Mapping (EIM) Guide and Reference*, SA22-7875, to understand what it takes to create your own EIM registry type.

7.3 The EIM client

EIM applications on z/OS must be APF authorized. Requiring APF authorization prevents inadvertent or malicious use of EIM APIs to change information in an EIM domain or to extract unauthorized information, which means that you must set the APF authorization extended attribute for each EIM application program residing in HFS files. This attribute is set by using the **`extattr`** command.

Important: As mentioned before, an EIM client is to perform lookup operations within a specific EIM Domain, for the purpose of retrieving a user ID to be presumably used for access control. It has also been said that EIM does not deal with user or entity authentication. The assumption is therefore that all lookup parameters are provided by trusted applications, which properly proceeded with user or entity authentication whenever it was necessary.

An application can perform one of two types of EIM lookup operations based on the type of information the application supplies as the source of the EIM lookup operation: A user identity or an EIM identifier.

The client application can use these calls, provided its identity is in the lookup access control group:

- eimGetTargetFromSource
- eimGetTargetFromIdentifier
- eimGetAssociatedIdentifiers

If successful, the lookup operation returns either an EIM Identifier or a user identifier in the specified registry.

The application supplies a user identity as a source

The application must also supply:

- The EIM registry definition name for the source user identity
- The EIM registry definition name that is the target of the EIM lookup operation

To be used as the source in a EIM lookup operation, a user identity must have a source association defined for it.

The application supplies an EIM identifier as a source

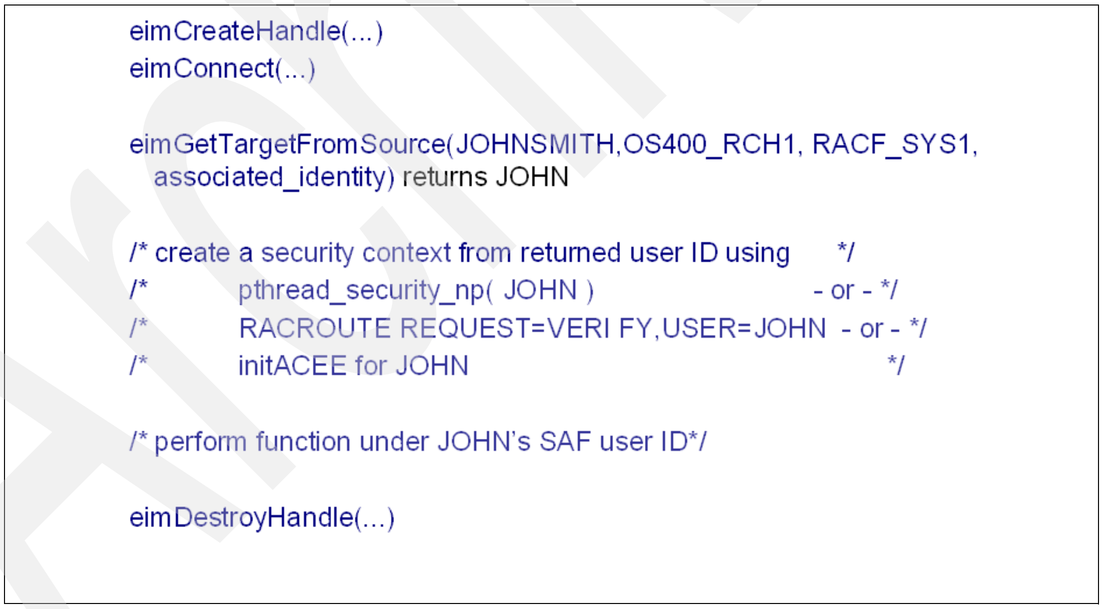
The application must also supply the EIM registry definition name that is the target of the EIM lookup operation.

For a user identity to be returned as the target of either type of EIM lookup operation, the user identity must have a target association defined for it.

The supplied information is passed to the EIM lookup operation that searches the EIM Domain Controller for the source association that matches the supplied information. Based on the EIM identifier (supplied to the API or determined from the source association information), the EIM lookup operation then searches for a target association for that identifier that matches the target EIM registry definition name, and provides application-specific information if any were defined.

Typically, the client application proceeds with a sequence of actions, as shown in Figure 7-8. In this example the application is executed on z/OS and has received a request from an OS400 system.

1. The eimCreateHandle() call allocates an EimHandle structure, which is used to identify the EIM connection during its life time and to maintain per-connection information. The EimHandle structure is passed on subsequent calls to other EIM operations. When the ldapURL parameter is NULL, EIM attempts to read the domain name, LDAP URL, and bind DN from RACF profiles, as described in 7.3.1, “Using RACF profiles to keep EIM default parameters” on page 174.
2. The eimConnect() call performs the bind to the LDAP Domain Controller and establishes the search tree base at the specified EIM domain. Connection information provided in the call specifies what kind of authentication is to be performed for the LDAP bind and whether the communication is protected with SSL. If the simple bind credentials in the eimConnectInfo structure are null, and connection information was obtained from RACF on the eimCreateHandle, a call to the R_DCEKEY callable service is made to decrypt the LDAP bind password stored in RACF.
3. The eimtargetgetFromSource() function in this example requests a mapping from identity JOHNSMITH in registry OS400_RCH1 to the corresponding identity in registry RACF_SYS1.
4. Once the z/OS local identity has been obtained, it can be used to be attached to the task, process, or thread serving the request (assuming that the client application has proper privilege to do so).
5. When done, the eimDestroyHandle() function frees resources associated with the EimHandle and closes connections to the EIM domain controllers.



```
eimCreateHandle(...)
eimConnect(...)

eimGetTargetFromSource(JOHNSMITH,OS400_RCH1, RACF_SYS1,
  associated_identity) returns JOHN

/* create a security context from returned user ID using      */
/*         pthread_security_np( JOHN )                 - or - */
/*         RACROUTE REQUEST=VERI FY,USER=JOHN  - or - */
/*         initACEE for JOHN                                  */

/* perform function under JOHN's SAF user ID*/

eimDestroyHandle(...)
```

Figure 7-8 Typical client application

7.3.1 Using RACF profiles to keep EIM default parameters

In this section we review using RACF profiles to keep EIM default parameters.

LDAP and domain bind information

Several profiles can be used to store in RACF an LDAP URL, the bind distinguished name, and the bind password, that can then be retrieved by the EIM services. The choice of the profile is dictated by the intent to either establish a system default that is used for EIM only or that can be used by other functions (such as z/OS PDAS or PKI Services), or to establish a default setup for a specific application user ID. These profiles are searched in the following order, as shown in Figure 7-9 on page 176, by EIM services that have been invoked without specifying the Domain Controller or domain in their parameter list:

1. Application user ID specific default

   The user-specific default is in a profile in the LDAPBIND class, with an arbitrary name that should also appear in the EIM segment of the USER profile. An example of definition of this profile is:

   ```
   RDEFINE LDAPBIND BUCKSDOMAIN +
   EIM(DOMAINDN('ibm-eimDomain=Bucks Domain.o=ibm.c=us')) +
   OPTIONS(ENABLE)) +
   PROXY(LDAPHOST(ldap://another.big.host) +
   BINDDN('cn=EIM Application Lookups') BINDPW('secret'))
   ```

   To establish this profile as a default for a specific user SERVERID, the EIM segment in the USER profile can be updated as follows:

   ```
   ALTUSER SERVERID EIM(LDAPPROF(BUCKSDOMAIN))
   ```

2. System-wide default

   The IRR.EIM.DEFAULTS profile in the LDAPBIND class establishes a default. An example of a definition of this profile is:

   ```
   RDEFINE LDAPBIND IRR.EIM.DEFAULTS +
   EIM(DOMAINDN('ibm-eimDomain=Joes Domain.o=ibm.c=us')) +
   OPTIONS(ENABLE)) +
   PROXY(LDAPHOST(ldap://some.big.host) +
   BINDDN('cn=EIM Lookup') BINDPW('secret'))T
   ```

3. System-wide default

   IRR.PROXY.DEFAULTS profile in the FACILITY class. An example of a definition of this profile is:

   ```
   RDEFINE FACILITY IRR.PROXY.DEFAULTS +
   EIM(DOMAINDN('ibm-eimDomain=Joes Domain.o=ibm.c=us')) +
   OPTIONS(ENABLE)) +
   PROXY(LDAPHOST(ldap://some.big.host) +
   BINDDN('cn=EIM Lookup') BINDPW('secret'))
   ```

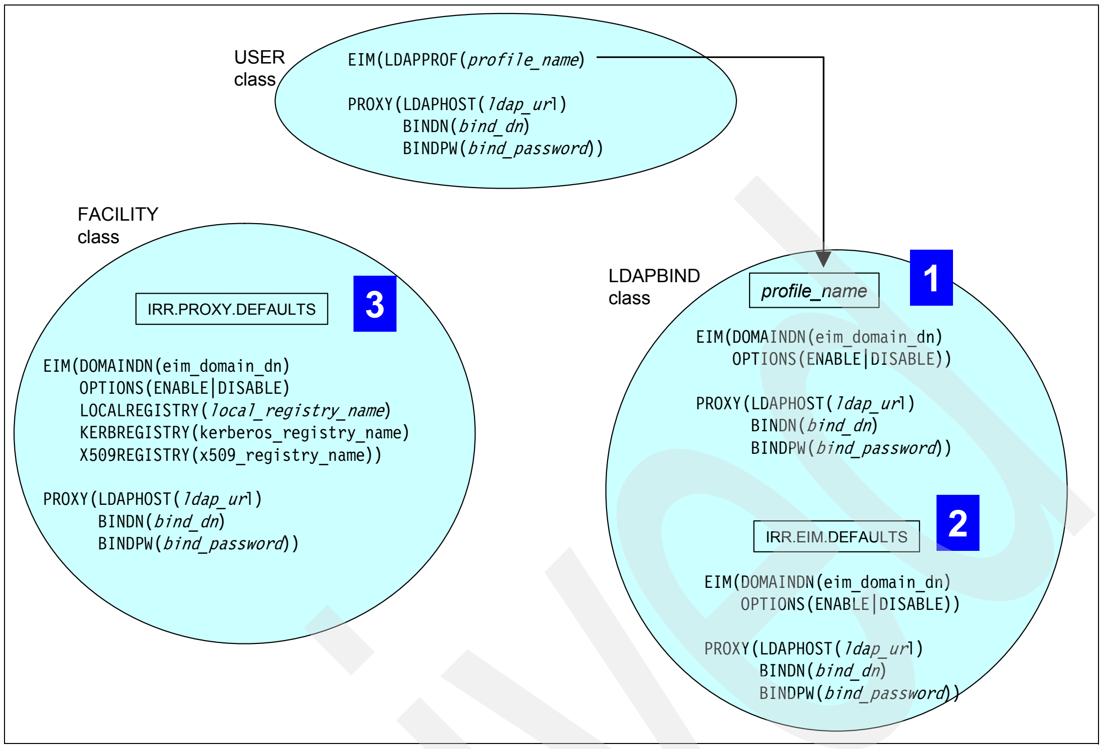


Figure 7-9 Keeping EIM LDAP and domain information in RACF profiles

The EIM and PROXY segment keywords and subkeywords combine to define the EIM domain, the LDAP host it resides on, and the bind information required by the EIM services to establish a connection with an EIM domain. The EIM services will attempt to retrieve this information when it is not explicitly supplied via invocation parameters.

Managing the local registry name

We are assuming here that the registry name is also specified in the EIM Domain Controller, presumably by using the eimadmin utility.

The local registry name can be set in the FACILITY IRR.PROXY.DEFAULTS profile by the RACF administrator for retrieval by the following functions, when the local registry is not indicated in the parameter list:

- eimGetTargetFromSource
- eimGetIdentifierFromSource
- eimGetAssociatedIdentifiers

The local registry name can be set with the following commands:

```
RALTER FACILITY IRR.PROXY.DEFAULTS EIM(LOCALREGISTRY('registry_name'))
SETROPTS EIMREGISTRY
```

Note: The SETROPTS command had to be used up to z/OS 1.5 to place the local registry name into storage so that it can be retrieved by applications. It is not necessary anymore to do a SETROPTS at z/OS 1.6.

A local registry name can be removed from storage with the following commands:

```
RALTER FACILITY IRR.PROXY.DEFAULTS EIM(NOLOCALREGISTRY)
SETROPTS EIMREGISTRY
```

Note: Same comment as above regarding the use of SETROPTS.

Kerberos and X.509 registries

z/OS 1.6 introduces the capability of setting defaults for a Kerberos registry or an X.509 certificate names registry. These defaults are set in the IRR.PROXY.DEFAULTS profile with the keywords KERBREGISTRY or X509REGISTRY.

The X509REGISTRY or KERBREGISTRY is the name of the registry that performed the authentication of the source user ID. These registries can be another name for the local RACF registry if RACF was used to authenticate an X509 certificate or a Kerberos principal. The registry names can also represent an x.509 registry or Kerberos registry that is not on z/OS.

7.4 A simple demonstration of EIM

To give you a better feel for what EIM is about, we have put together a simple Web application that performs an EIM lookup operation. The following sections describe the demonstration and how to set it up.

7.4.1 The eimdemo application

This Web application is intended to display information about employees in the enterprise. The administrator needs to log on using her RACF user ID and password and then enter the name of the employee (actually an EIM identifier). The information that is displayed belongs to the employee and has to be accessed using the employee's user ID.

The demonstration requires the following components:

- A Web browser on the administrator's workstation
- An HTTP server
- A static Web page
- Two C programs that retrieve the information about the employee and create the dynamic Web page that displays the results
- A file in the local directory of the employee containing the employee's personal information
- An EIM domain to host the mapping information between the employees' names and their user IDs

The demo environment is shown in Figure 7-10 on page 178. Three users have been registered in RACF, the target registry, as:

- YURY, for the EIM identifier Yury Kritchever
- PEGGY for Peggy Labelle
- JON for Jon Briggs

Note also that these users do have a UID in their RACF OMVS segment and their personal data are residing in the HFS file with a path of the form /u/<racf_userid>/test.data with permission bits for owner access only (700).

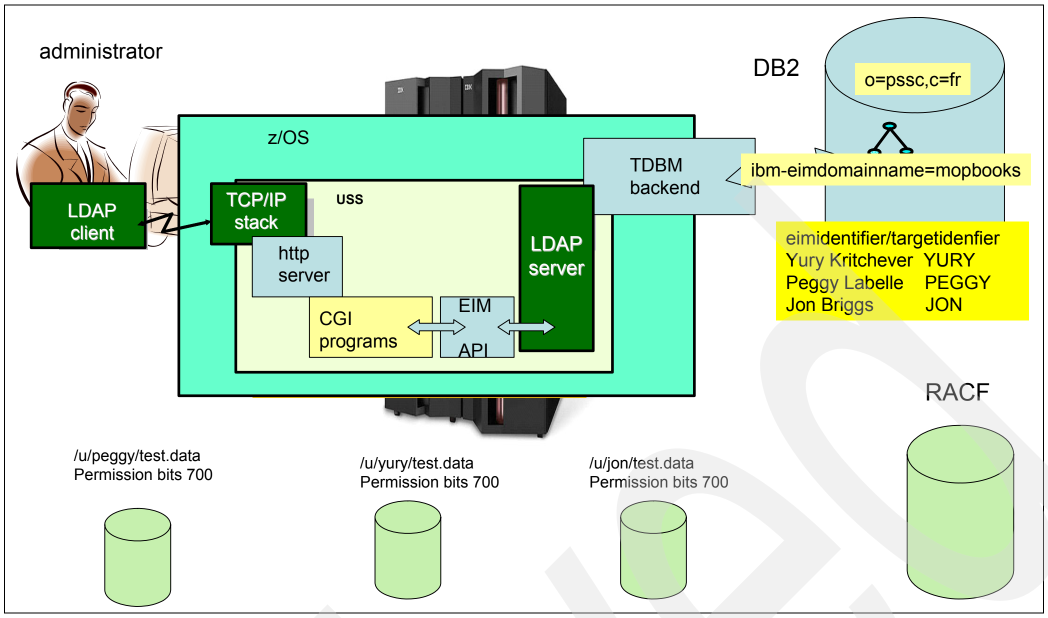


Figure 7-10 The demo environment

The demo flow

The demo steps are shown in Figure 7-11 on page 179. The demo steps are:

1. The administrator calls the eimdemo Web site and is prompted to enter a user ID and password, as the EIM demo URL is protected.
2. After successful authentication of the administrator, the Web page shown in Figure 7-12 on page 179 is presented so that the administrator can fill the EIM identifier field, in this example with the EIM identifier Peggy LaBelle. Posting this value by clicking the **Access Data** button triggers a CGI program, running, in this example, with the RACF user ID DEMOSRV.
3. The CGI main program spawns a child process, running with the same DEMOSRV identity, which is to perform the lookup in the EIM Domain Controller, running the typical client sequence shown in Figure 7-8 on page 174, which comprises an eimGetTargetFromIdentifier() function.
4. The Domain Controller returns the user ID in the target registry, that is, the RACF user ID.
5. A new child program is then spawned with an identity switch and runs under the RACF user ID returned in the previous step. That is, it runs with the UID of the owner of the personal data file, and builds the access path to this file. It eventually reads the personal data and returns them to the parent process.
6. The main CGI program then composes a Web page with the returned data and sends it to the requestor. An example of a response is given in Figure 7-13 on page 180. In this example the personal data file had just one line of text: `Hello, I'm Peggy....`

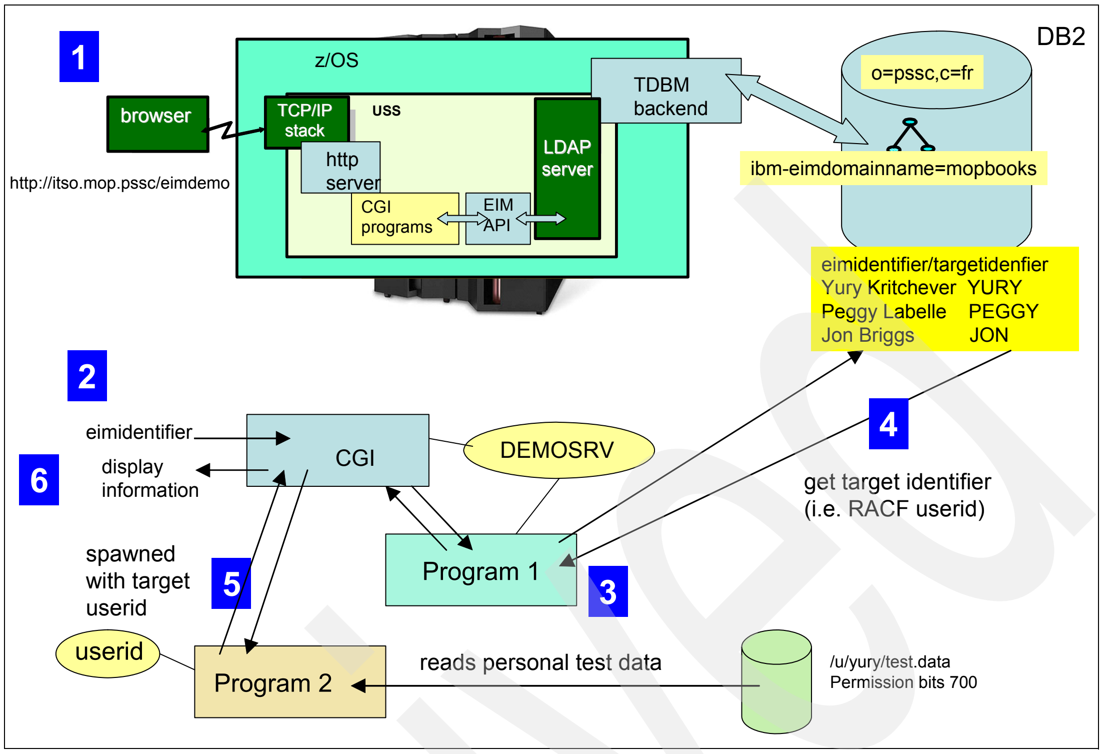
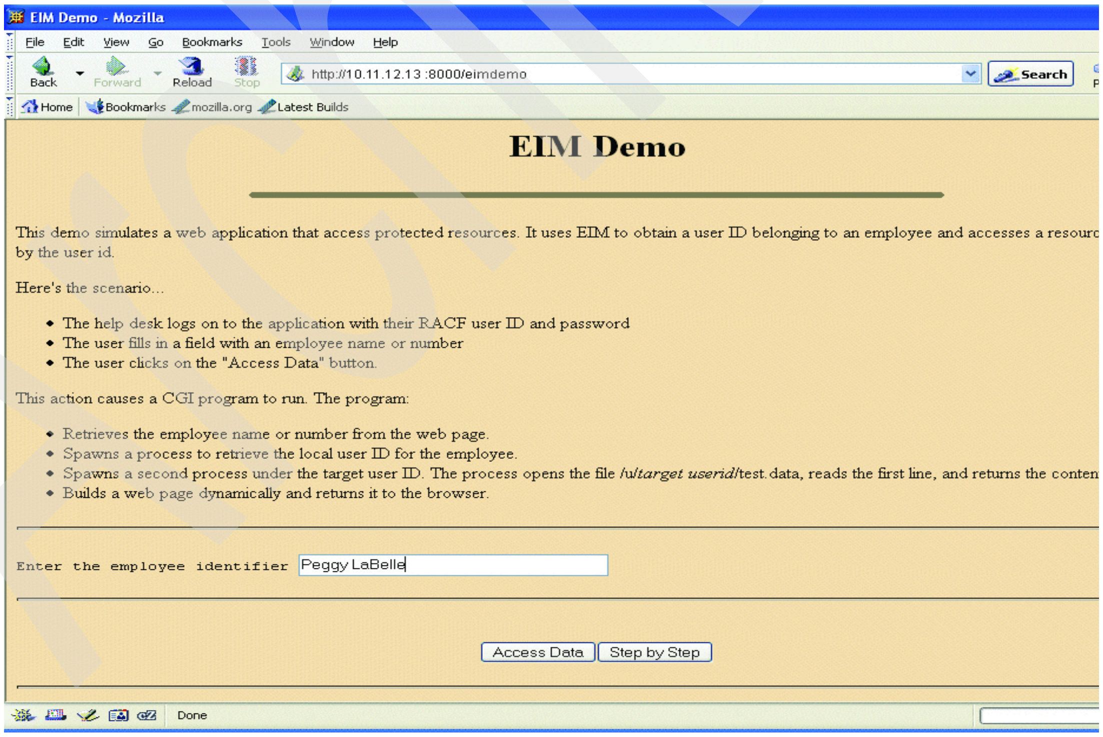


Figure 7-11 Demo flow


EIM Demo - Mozilla
File Edit View Go Bookmarks Tools Window Help
Back Forward Reload Stop
http://10.11.12.13 :8000/eimdemo
Search
Home Bookmarks mozilla.org Latest Builds
Access Data
Step by Step
Done

EIM Demo

This demo simulates a web application that access protected resources. It uses EIM to obtain a user ID belonging to an employee and accesses a resourc
by the user id.

Here's the scenario...

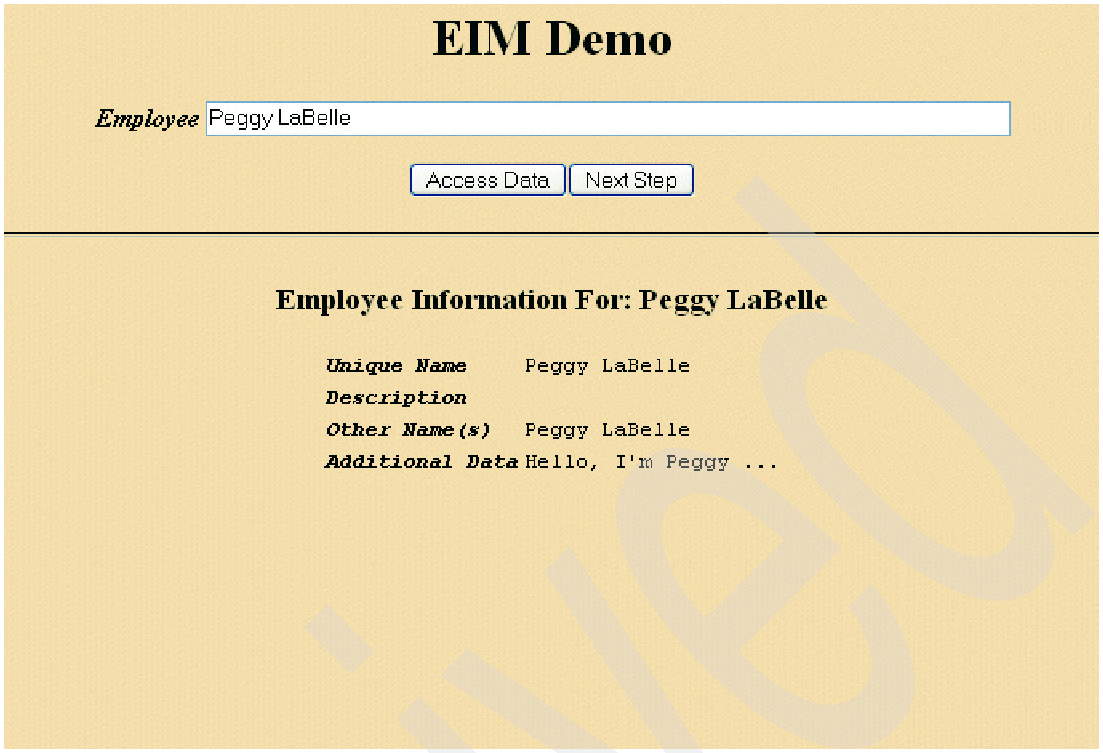
- The help desk logs on to the application with their RACF user ID and password
- The user fills in a field with an employee name or number
- The user clicks on the "Access Data" button.

This action causes a CGI program to run. The program:

- Retrieves the employee name or number from the web page.
- Spawns a process to retrieve the local user ID for the employee.
- Spawns a second process under the target user ID. The process opens the file /u/*target userid*/test.data, reads the first line, and returns the conten
- Builds a web page dynamically and returns it to the browser.

Enter the employee identifier Peggy LaBelle

Figure 7-12 The EIM demo welcome page



Employee Information For: Peggy LaBelle

```
Unique Name      Peggy LaBelle
Description
Other Name(s)    Peggy LaBelle
Additional Data  Hello, I'm Peggy ...
```

Figure 7-13 The EIM demo response page

7.4.2 Sample code

The sample code we used is available in Appendix A, "EIM API demo sample code" on page 263. The main CGI is the eimdemo.c program, the first child is eimdemo2.c, and the second child is eimdemo3.c.

7.4.3 Setup of the demo environment

We are assuming here that the reader already has z/OS http and LDAP servers running.

z/OS LDAP server specifics

The EIM schemas must have been loaded. They are delivered in z/OS in the file /usr/lpp/ldap/etc/schema.IBM.ldif.

In our demo we are using an LDAP administrator identity, with a DN of "cn=ldapadmin" and a password *secret*. These two parameters have been set in the LDAP configuration file (slapd.conf) with the directives adminDN and adminPW; that is, we did not bother to have an LDAP entry for the administrator *person* object.

We are using the basic LDAP server configuration; that is, no replica, no referrals, no multi-server environment.

Our LDAP server is running at IP address 10.11.12.13, listening on port 389.

z/OS http server specifics

We have organized the files as follow:

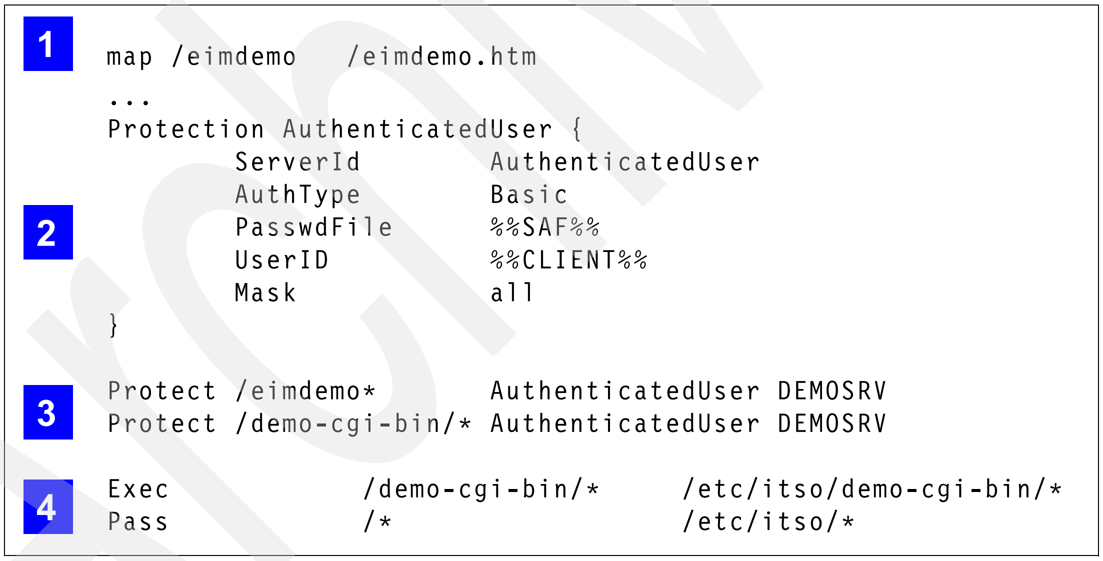
- The demo static Web page shown in Figure 7-12 on page 179 is located at /etc/itso/eimdemo.html.

- The demo executable code is stored in:
  - /etc/itso/demo-cgi-bin/eimdemo
  - /etc/itso/demo-cgi-bin/eimdemo1
  - /etc/itso/demo-cgi-bin/eimdemo2

  All with the APF and program extended attributes turned on by extattr +a and extattr +p commands.

The http server configuration files has been set up with the directives shown in Figure 7-14.

1. A *map* directive, along with the *pass* directive in 4, allows you to use a shortcut to the static welcome page.
2. The protection directive AuthenticatedUser requests the user to enter a user ID and password, which are eventually validated by RACF. Note, however, that the mask directive, still for simplicity, is left to *All*, meaning that there is no additional filtering for authentication on top of RACF authentication.
3. The AuthenticatedUser protection directive is invoked for URLs pointing at eimdemo or demo-cgi-bin. The trick here is that even after authentication of the requesting user by RACF, the Web server thread will be running with the user ID DEMOSRV, as forced by the syntax of the Protect directive.
4. These are the usual exec and pass directives indicating that a directory contains executables, and providing a catch-all path for URLs that fell through the configuration file directives.



```
1   map /eimdemo   /eimdemo.htm
    ...
    Protection AuthenticatedUser {
            ServerId        AuthenticatedUser
            AuthType        Basic
2           PasswdFile      %%SAF%%
            UserID          %%CLIENT%%
            Mask            all
    }

3   Protect /eimdemo*       AuthenticatedUser DEMOSRV
    Protect /demo-cgi-bin/* AuthenticatedUser DEMOSRV

4   Exec            /demo-cgi-bin/*     /etc/itso/demo-cgi-bin/*
    Pass            /*                  /etc/itso/*
```

Figure 7-14 Demo httpd.conf file

RACF specifics

RACF has been set up with user profiles for PEGGY, YURY, and JON—every one of them with a unique UID>0 in an OMVS segment.

- The DEMOSRV user ID has UID(0) and has been set up with an EIM segment pointing to the EIMDEMO profile in the LDAPBIND class with the following command:

  ```
  ALU DOMSRV EIM(LDAPPROF(EIMDEMO))
  ```

Using RACF profiles for default EIM parameters

As explained in 7.3.1, “Using RACF profiles to keep EIM default parameters” on page 174, we can use RACF profiles to store parameters needed to bind to the EIM Domain Controller and to specify a default EIM domain. We have set up the following defaults:

- In the IRR.PROXY.DEFAULTS profile in the FACILITY class:

  ```
  RDEFINE FACILITY IRR.PROXY.DEFAULTS EIM(LOCALREGISTRY(MVO8RACF))
  SETR RACLIST(FACILITY) REFRESH
  ```

- In the EIMDEMO profile the LDAPBIND class. Remember that the EIMDEMO profile is pointed at in the EIM segment of the DEMOSRV user profile. The LDAPBIND profile has been created using the following command:

  ```
  RDEFINE LDAPBIND EIMDEMO PROXY(LDAPHOST(LDAP://10.11.12.13:389) +
  BINDDN('cn=demosrv,o=moppssc,c=fr') BINDPW('secret') +
  EIM(DOMAINDN(‘ibm-eimdomainname=MopBooks,o=pssc,c=fr’))
  ```

  Note that in order for the BINDPW value to be processed by RACF, a profile LDAP.BINDPW.KEY must have been defined in the KEYSMSTR class. To define this profile we issued the command:

  ```
  RDEFINE KEYSMSTR LDAP.BINDPW.KEY + SSIGNON(KEYMASKED(0023428875DECFAC))
  ```

Set up of the EIM domain

The eimadmin utility has been used to set up the EIM domain.

Creating the domain

The following eimadmin shell command has been used (note that the command is a single line, or can be split with continuation characters) to define the EIM domain MopBooks:

```
eimadmin -aD -d "ibm-eimdomainname=MopBooks,o=pssc,c=fr"
-h ldap://10.11.12.13:389 -b cn=ldapadmin -w secret
```

Defining an LDAP entry for the EIM administrator

An LDIF file called eimadmin.ldif has been prepared that contains:

```
dn: cn=eim administrator,o=moppssc,c=fr
objectclass: top
objectclass: person
sn: eim administrator
cn: eim administrator
userpassword: secret
```

Then the LDAP directory has been updated with the command:

```
ldapadd -h ldap://10.11.12.13 -D cn=ldapadmin -w secret -f eimadmin.ldif
```

Adding the EIM administrator to the administration access group

This is done by invoking the eimadmin utility with:

```
eimadmin -aC -d 'ibm-eimDomainName=MopBooks,o=moppssc,c=fr' \
-q 'cn=eim administrator,o=moppssc,c=fr' -f DN \
-c admin -h ldap://10.11.12.13:389 -b 'cn=ldapadmin' -w secret
```

Adding a registry

The registry MVO8RACF with type RACF has been added with the command:

```
eimadmin -aR -r MVO8RACF -y RACF \
-d "ibm-eimdomainname=MopBooks,o=pssc,c=fr" -h ldap://10.11.12.13:389 \
-b ‘cn=eim administrator’ -w secret
```

Adding EIM identifiers

Three identifiers have been added with the following commands:

```
eimadmin -aI -i 'Jon Briggs'      -o 'Troubleshooter'                  \
-d "ibm-eimdomainname=MopBooks,o=pssc,c=fr" -h ldap://10.11.12.13:389 \
-b cn=’eim administrator’ -w secret

eimadmin -aI -i 'Peggy LaBelle' \
-d "ibm-eimdomainname=MopBooks,o=moppssc,c=fr" -h ldap://10.11.12.13:389 \
-b ‘cn=eim administrator’ -w secret

eimadmin -aI -i 'Yury Kritchever'                                        \
-d "ibm-eimdomainname=MopBooks,o=moppssc,c=fr" -h ldap://10.11.12.13:389 \
-b ‘cn=eim administrator’ -w secret
```

Adding associations

Since the lookup required by the demo is based on the provided EIM identifier, only a target association is needed. The following commands associate the identifiers to user IDs in a RACF type registry.

```
eimadmin -aA -i 'Jon Briggs'      -r MVO8RACF -u Jon   -t target       \
-d "ibm-eimdomainname=MopBooks,o=moppssc,c=fr" -h ldap://10.11.12.13:389 \
-b ‘cn=eim administrator’ -w secret

eimadmin -aA -i 'Peggy LaBelle'   -r MVO8RACF -u Peggy -t target       \
-d "ibm-eimdomainname=MopBooks,o=moppssc,c=fr" -h ldap://10.11.12.13:389 \
-b ‘cn=eim administrator’ -w secret

eimadmin -aA -i 'Yury Kritchever' -r MVO8RACF -u Yury  -t target \
-d "ibm-eimdomainname=MopBooks,o=moppssc,c=fr" -h ldap://10.11.12.13:389 \
-b ‘cn=eim administrator’ -w secret
```

Creating an LDAP entry for an EIM client

We have prepared an LDIF file, called eimuser.ldif, which contains:

```
DN: cn=demosrv,o=moppssc,c=fr
objectclass: top
objectclass: person
sn: eim user
cn: eim user
userpassword: secret
```

The LDAP entry is created with:

```
ldapadd -h ldap://10.11.12.13 -D cn=ldapadmin -w secret -f eimuser.ldif
```

Adding the EIM client to the mapping access group

The commands used are:

```
eimadmin -aC  -d 'ibm-eimDomainName=MopBooks,o=moppssc,c=fr' \
-q 'cn=demosrv,o=moppssc,c=fr' -f DN \
-c mapping -h ldap://10.11.12.13:389 -b 'cn=eim administrator' -w secret
```

7.5 New EIM features at z/OS 1.6

z/OS 1.6 introduces the following features:

- Optional default mappings policies for certificates, registries, and domains
- Definition of a registry of names from digital certificates

- An updated eimadmin utility.
- The capability of configuring in z/OS default registry names for Kerberos and X509 registries (already mentioned in 7.3.1, “Using RACF profiles to keep EIM default parameters” on page 174)

Mapping policies

As seen above, EIM mapping is based on specific identifiers in the EIM domain. These specific associations can be complemented by default associations, that is, policy associations, which apply to users without an association to a specific EIM identifier. The intent of these default associations is to have many users who are not mapping to a specific identifier, mapping to the same default identifier, thus bringing simplification to the administration of large numbers of users.

Policy association can be expressed for distinguished names in certificates, for domains, and for registries. The policy associations have to be explicitly enabled before being taken into account in lookup operations. A lookup operation is looking for the following, in this order:

1. Exact mapping via an EIM identifier
2. Name found in the X509 Registry - Apply certificate name policy if defined
3. Registry policy
4. Domain policy

7.5.1 Registry and domain mapping policies

Here we explain the setup corresponding to the associations shown in Figure 7-15 on page 185. The regular associations are set up with:

- Source association from REGA: John Smith to identifier: John N. Smith

  ```
  eimadmin –aA -t SOURCE
  -r REGA –u ‘John Smith’ –i’John N. Smith’
  –h ldap host name –d EIM domain name
  –b bind DN –w password
  ```

- Target association from identifier John N. Smith to REGB: JSMITH:

  ```
  eimadmin –aA -t TARGET
  –i’John N. Smith’ –r REGB –u JSMITH
  –h ldap host name –d EIM domain name
  –b bind DN –w password
  ```

We now set up a registry policy stating that a lookup from REGA to REGB that does not find a specific identifier or certificate name association returns the user ID EMPLOYEE in REGB:

```
eimadmin –aY -t REGISTRY
-r REGA –T REGB –u EMPLOYEE
–h ldap host name –d EIM domain name
–b bind DN –w password
```

The next example is a domain policy stating that any user in the domain, with a target registry of REGC, that does not find any association maps to the user ID UNKNOWN in REGC.

```
eimadmin –aY -t DOMAIN
--r REGA –T REGC –u UNKNOWN
–h ldap host name –d EIM domain name
–b bind DN –w password
```

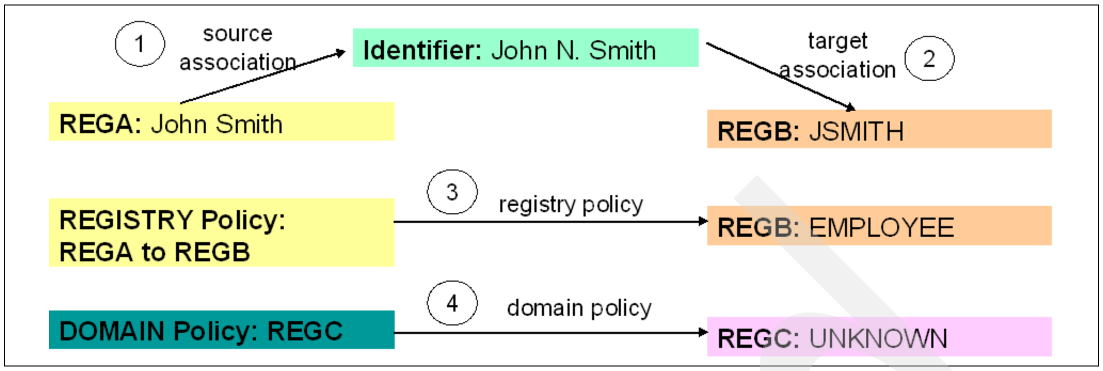


Figure 7-15 Registry and domain policies

7.5.2 Digital certificate registries

Prior to z/OS 1.6, installations can create their own custom registry for storing associations between the subject's DN and the issuer's DN from a certificate and an EIM identifier.

Beginning with In z/OS 1.6, EIM administrators can create X509 type registries, with EIM client functions to assist with extracting the SDN or IDN from a certificate so that they can be used as a source identity.

This is augmented by the capability of defining a certificate filter policy, as shown in Figure 7-16 on page 186.

Associations with an X509 registry

The following eimadmin command associates a source certificate, JSmith.cert, in the X509 registry X509REGA to the identifier John N Smith:

```
eimadmin –aA -t SOURCE
-r X509REGA –E JSmith.cert –i'John N. Smith'
–h ldap host name –d EIM domain name
–b bind DN –w password
```

Actually, the association is built using the certificate subject's DN, which is 'cn=John Smith,o=ibm,c=us' in this example, and the issuer's DN 'o=cert auth,c=us'.

Certificate filter between X509REG and REGB, where the source user certificate subject's DN is 'cn=John Smith,o=ibm,c=us' and the issuer's DN is 'o=cert auth,c=us'.

Creating a certificate filter policy

A certificate filter policy is to specify a subset of certificates in the source registry, that are to map to a default target registry user ID if no specific identifier association can be found.

Here is an example where certificates in the X509 registry X509REGA are to map the default target user ID IBMEMP in registry REGB if no specific identifier association can be found. In this example, the source certificates candidate for this default mapping should have a subject's DN ending with "o=ibm,c=us", and an issuer's DN of "o=cert auth,c=us'.

The **eimadmin** command to establish this certificate filter policy is:

```
eimadmin –aY -t FILTER
-r X509REGA
–J 'o=ibm,c=us'-F 'o=cert auth,c=us'
```

```
-T REGB –u IBMEMP
–h ldap host name –d EIM domain name
–b bind DN –w password
```

Where -t is the certificate filter policy, -r is the source registry name, -J is the SDN, and -F is the IDN filter.

Figure 7-16 also shows that if the certificate filter policy cannot be applied, then the EIM lookup checks if a registry policy can be used; if not, then it checks if a domain policy can fit the lookup request.

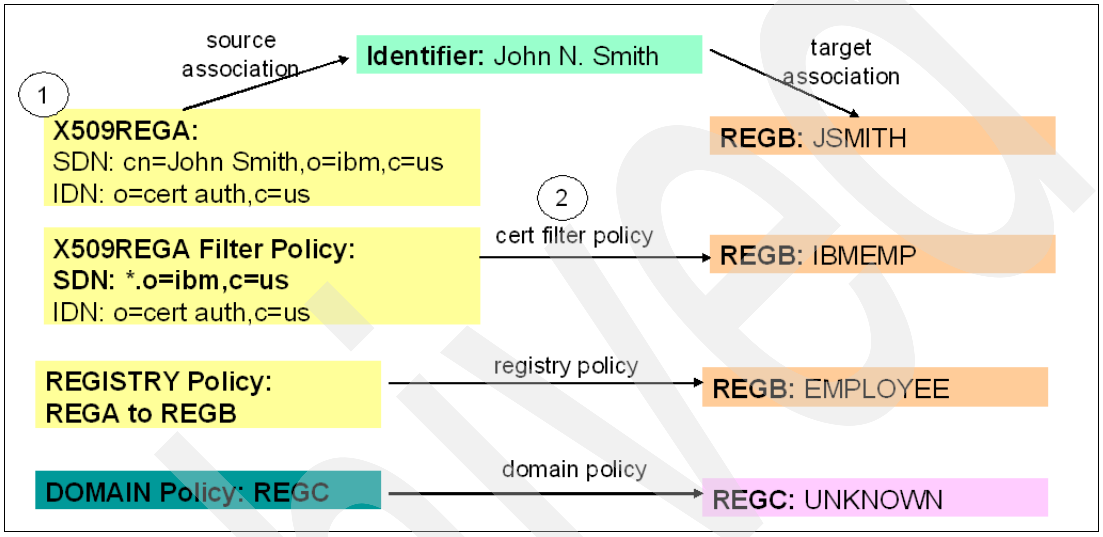
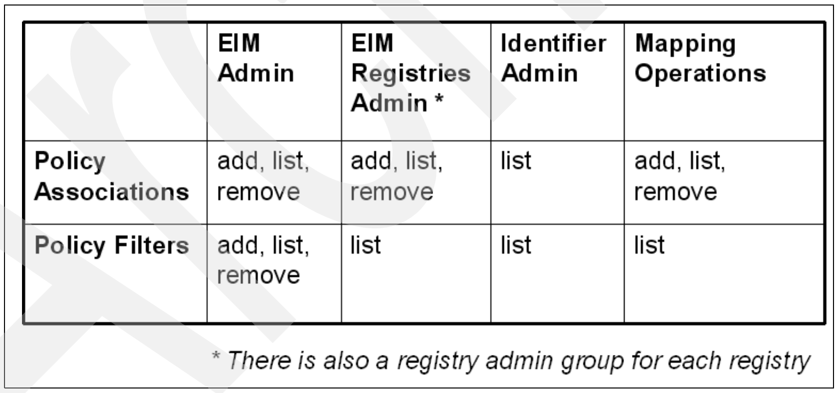


Figure 7-16 Certificate filter policy example

The specific access controls to policies are shown in Figure 7-17.



	EIM Admin	EIM Registries Admin *	Identifier Admin	Mapping Operations
Policy Associations	add, list, remove	add, list, remove	list	add, list, remove
Policy Filters	add, list, remove	list	list	list

** There is also a registry admin group for each registry*

Figure 7-17 Policies access controls

8

z/OS Network Authentication Service (Kerberos)

Network Authentication Service is a z/OS implementation of the Kerberos V5 protocol. It has been made available as a z/OS standard feature beginning with OS/390 2.10. The information provided here complements a former description of the Network Authentication Service in the redbook *Putting the Latest z/OS Security Features to Work*, SG24-6540, which also explains how to set up this service.

This chapter focuses on the enhancements introduced into Network Authentication Service since z/OS 1.2 up to z/OS 1.6.

Additionally, a Kerberos test program is now provided in z/OS Network Authentication Service. We explain here how to set up and run this test.

8.1 A brief reminder on z/OS Network Authentication Service

Figure 8-1 on page 189 is an overall view of the z/OS Network Authentication Services. It is a z/OS implementation of a Kerberos Key Distribution Center (KDC). There are three main players in this implementation:

- A z/OS UNIX application, started as the started task SKRBKDC that acts as:
  - An Authentication Service (AS) server that provides Kerberos Authentication Tickets (also called TGT for Ticket Granting Ticket) to TCP/IP client.
  - A Ticket Granting Service (TGS) that provides Kerberos Service Tickets.
- RACF, or an equivalent product, acting as a Kerberos registry that the SKRBKDC server or a Kerberos-enabled application invoke, through the SAF interface, with RACF callable services—the Kerberos registry information being themselves stored in RACF profiles, and are being maintained using RACF commands.
  - The RACF USER profiles have been extended with a KERB segment to hold the information pertaining to the Kerberos identity of the user, that is, the Kerberos principal name, along with the user's Kerberos DES key.
  - New profiles have been created for Kerberos use:
    - Profiles in the REALM class define the Kerberos realm that the RACF registry covers, and optionally other realms that are trusted by the RACF KDC. The RACF local realm profile holds the realm's Kerberos DES key, along with policy information.
    - Profiles in the KERBLINK classes that contain information to map a Kerberos principal name to a RACF user ID.

Note: The USER and REALM profiles hold the user's or realm's secret DES key. This key is actually derived from the user's or realm's RACF password value.

Note: The RACF Kerberos registry can be shared in a parallel sysplex environment where multiple instances of the SKRBKDC server are running in the sysplex members.

- A Kerberos-enabled application that accepts Kerberos tickets as authentication credentials, and which, optionally, use the Kerberos session key to encrypt the transferred data. The application interacts with the SAF Kerberos registry using RACF callable services. Note that the application can also request the local RACF KDC to map a Kerberos principal to a RACF user ID.

Note: z/OS Network Authentication Service also provides the GSS and krb5_ APIs for Kerberos-enabled applications.

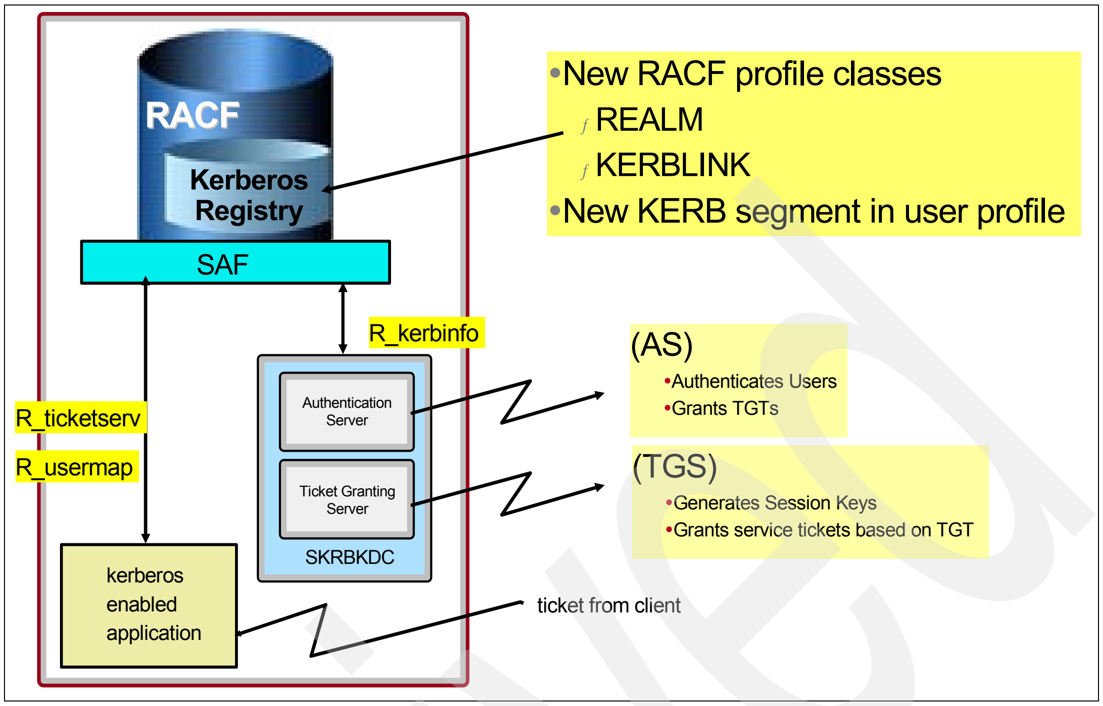


Figure 8-1 z/OS Network Authentication Service

The details of the z/OS Network Authentication Service operations and setup can be found in:

- The redbook *Putting the Latest z/OS Security Features to Work*, SG24-6540
- In the z/OS reference books:
  - *z/OS Integrated Security Services Network Authentication Service Administration*, SC24-5926
  - *z/OS Integrated Security Services Network Authentication Service Programming*, SC24-5927
  - *z/OS Security Server RACF Security Administrator's Guide*, SA22-7683
  - *z/OS Security Server RACF Callable Services*, SA22-7691

The SKRBKDC z/OS UNIX application

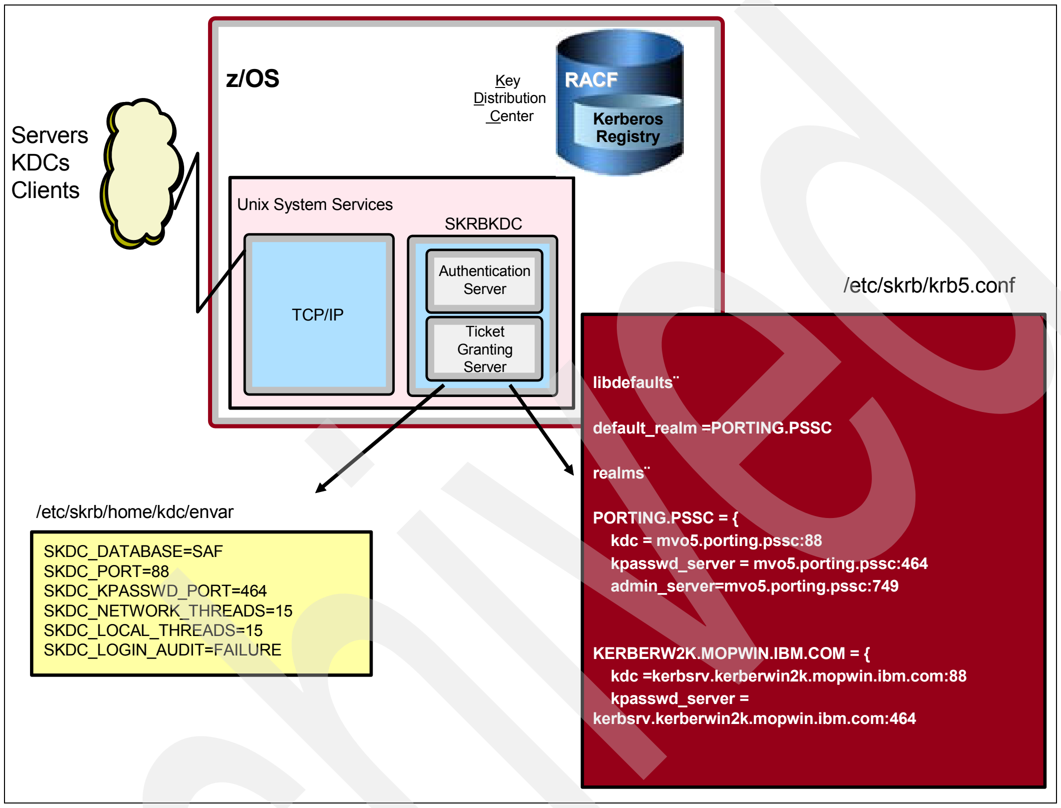
Figure 8-2 on page 190 is a schematic view of the SKRBKDC server, with the two runtime files it uses:

- The krb5.conf file - Where information such as the KDCs in the Kerberos configuration are specified. In our example there are two KDCs in the configuration:
  - The KDC located at IP address mvo5.porting.pssc, which covers the Kerberos realm PORTING.PSSC. This KDC happens to be the one where the SKRBKDC server is running, hence the specification "default_realm=PORTING.PSSC" in the configuration file.
  - The KDC located at IP address kerbsrv.kerberwin2k.mopwin.ibm.com, which covers the KERBERW2K.MOPWIN.IBM.COM realm. Having specified this KDC in the configuration file allows the SKRBKDC to interact with this other KDC ID needed.

  Note that for the both KDCs, Kerberos tickets can be obtained via requests to port 88, and passwords in the Kerberos registry can be changed by requests to port 464.

Other information, not shown here, in the configuration file relate to such things as the KDC policy regarding tickets lifetime.

- The envar file, where environment variables are used to specify to the SKRBKDC server parameters such as the registry type (SAF in this example), the ports the server has to bind to, etc.



```
SKDC_DATABASE=SAF
SKDC_PORT=88
SKDC_KPASSWD_PORT=464
SKDC_NETWORK_THREADS=15
SKDC_LOCAL_THREADS=15
SKDC_LOGIN_AUDIT=FAILURE
```

```
libdefaults¨

default_realm =PORTING.PSSC

realms¨

PORTING.PSSC = {
   kdc = mvo5.porting.pssc:88
   kpasswd_server = mvo5.porting.pssc:464
   admin_server=mvo5.porting.pssc:749


KERBERW2K.MOPWIN.IBM.COM = {
   kdc =kerbsrv.kerberwin2k.mopwin.ibm.com:88
   kpasswd_server =
kerbsrv.kerberwin2k.mopwin.ibm.com:464
```

Figure 8-2 The SKRBKDC UNIX server

Kerberos commands

UNIX implementations of Kerberos are supporting a set of commands for Kerberos clients and administrators. Although not standardized, they look very much the same either in pyre UNIX or other platforms' implementations, with slight differences in their syntax or operational behavior. z/OS Network Authentication Service provides such a set of Kerberos commands. They are, in the initial implementation at OS/390 2.10:

- kinit: Obtains the initial Ticket Granting Ticket
- klist: Displays the list of tickets obtained so far and kept in a credentials cache
- kdestroy: Destroys all obtained tickets kept in a credentials cache
- keytab: Allows to manage a local key table
- ksetup: Manages Kerberos service entries in the LDAP directory for a Kerberos realm

8.2 z/OS Network Authentication Service enhancements

In this section we review z/OS Network Authentication Service enhancements.

8.2.1 z/OS 1.2

Here we review those for z/OS 1.2.

Kerberos principals and realms' DES key in the RACF registry

The Kerberos DES key can be specified on an individual profile basis, that is, in USER or REALM profiles, as:

- DES - That is the initial implementation.
- Triple DES - The user's or realm's Kerberos key is a triple-DES key.
- DES with Derivation - The user's or the realm's key is considered to be a *base key*. The actual key used is a DES key, of which the value depends on the type of Kerberos message needing a key, and that is, however, derived from the base key. This provides for additional security when using single DES keys.

The key type is defined in the KERB segment of the USER or REALM profile and is specified as ENCRYPT(DES|DES3|DESD), or, to prohibit a key type, as ENCRYPT(NODES|NODES3|NODESD). For example, the following command allows the user YURY to use a user Triple-DES Kerberos key:

```
ALU YURY KERB(ENCRYPT(DES3))
```

In order to control the use of the capability of using keys that are not plain DES keys, it is required that the RACF administrator sets the option KERBLVL(1):

```
SETR KERBLVL(1)
```

Not setting this option or setting KERBLVL(0) results in always generating DES keys. You may want this to happen if not all of your platforms in your Kerberos configuration support Triple-DES or derived keys.

Notes: The three keys are always generated and stored as a user key when KERBLVL is set to "1". The settings of ENCRYPT will determine which keys are allowed for use. If ENCRYPT is changed then there is no need to generate new keys since they are already available in the USER profile.

Kerberos Server enhancements

Kerberos Server enhancements are:

- Triple-DES is supported with proper changes in the server's configuration file, the GSS-API, and the UNIX commands.
- The Kerberos server invokes ICSF, if in operation on the system, to proceed with hardware DES/Triple-DES encryption and decryption.
- The following new Kerberos commands are made available:
  - The kadmin command - For miscellaneous administration of the Kerberos principals in a non-RACF KDC (the RACF KDC is still administered via RACF commands).
  - kpasswd - To change principal's password (via the Kerberos Password Server). Note that this command also changes the password in the RACF database.
  - kvno - To display a key version number.
- The server has a dynamic TCP/IP support, in that is does not fail when TCP/IP becomes unavailable. Likewise, it supports the dynamic addition of new TCP/IP network interfaces when a new TCP/IP stack is started or when a new TCP/IP device interface is activated.
- New parameters for the MODIFY SKRBKDC system console command:
  - Display contents of credentials' cache data space:

    ```
    DISPLAY CREDS,owner,date
    ```

  - Display active security servers within the sysplex:

    ```
    DISPLAY XCF
    ```

  - Display list of available encryption types, whether hardware crypto is available, and whether encryption may be used for user data:

    ```
    DISPLAY CRYPTO
    ```

  - Display the service level of Kerberos security server:

    ```
    DISPLAY LEVEL
    ```

- The SKRBKDC server can register to ARM:
  - The element type is SYSKERB - restart level 2.
  - The element name is EUVFKDC_sysname.
- kmigrate utility - This is a utility to help prime Kerberos registry from DCE or from RACF, by generating the proper ADDUSER and ALTUSER commands. It is a free download from:

  http://www.s390.ibm.com/racf/goodies.html

8.2.2 Network Authentication Service enhancement at z/OS 1.4

In this section we review the Network Authentication Service enhancement at z/OS 1.4.

Kerberos Server enhancements

Here we discuss the Kerberos Server enhancements.

The NDBM user registry

In order to better integrate the z/OS Network Authentication Service into an existing UNIX-like Kerberos environment, it was decided to add support for a second type of the Kerberos registry: The New data Base Manager (NDBM) registry. The NDBM behaves in a more familiar way to people with just UNIX-related skills and does provide more inter operability, administration-wise, with non-z/OS platforms. The NDBM registry resides in HFS or z/FS files. It is created and primed using Kerberos commands that are explained later in this chapter.

If we compare the RACF based Kerberos registry and the NDBM registry, each type of registry has its own advantage:

- NDBM pro’s
  - The KDC maintains its own registry database using the Unix System Services NDBM support. There are actually no dependencies on RACF skills and command interfaces.
  - Full Kerberos administration support is provided via z/OS UNIX commands and APIs.
  - Realm can contain z/OS KDC and non-z/OS KDC instances, with the capability of propagating registry data between these instances.
- NDBM limitations
  - The HFS files require their own protection and backup schemes.
  - The NDBM database is not sharable within a parallel sysplex.
  - There is no mapping of Kerberos principals to RACF user IDs with NDBM alone.
  - The NDBM registry is limited by the maximum size of an HFS database.
  - The propagation mechanism is inefficient for a large number of principals.
- RACF registry pro’s

  - The Kerberos information is integrated with the z/OS local user registry, using common administration and maintenance procedures, still falling under the responsibility of the RACF administrator.
  - The registry can be shared within a parallel sysplex.
  - The Kerberos-enabled applications are running with known RACF user IDs that eliminate the necessity to use Kerberos passwords and key tables when obtaining and decrypting tickets.
  - It scales well to support a large number of principals.
- RACF registry limitations
  - The Kerberos principals must be first defined as z/OS user IDs.
  - No actual UNIX-like Kerberos administration support is provided due to semantic differences between the RACF database and the Kerberos administration wire protocols.
  - All RACF profiles related to Kerberos must be kept synchronized across all KDC instances in the realm. Whenever possible the proper approach to achieve this is to have all KDC instances sharing the same RACF database.

Important: An NDBM registry does not provide mapping information between Kerberos principal names and RACF user ID. If such a mapping is desired (and it can then be exploited using the R_usermap RACF callable service) it takes defining in the RACF USER profiles the proper KERB information that matches the contents of the NDBM registry. If, for example, the NDBM registry contains the principal yuryprinc, to be able to map yuryprinc to the RACF user YURY, a KERB segment must be defined for the RACF USER profile YURY as:

```
ALU YURY KERB(KERBNAME(yuryprinc))
```

Two new Kerberos commands are introduced:

- kdb5_ndbm: This command invokes the Kerberos NDBM database maintenance utility. It is used to maintain a z/OS Kerberos NDBM registry and, for example, is capable of dumping the registry in a format compatible with the reference implementation, that is, with the MIT's Kerberos **`kdb5_util`** command. This option may be found very useful by those who wish to copy the Kerberos registry from one operating system platform to another.
- kpropd: This is the Kerberos stand-alone database propagation utility. It is used to perform a stand-alone database propagation from the primary KDC for the realm. This is usually done for creating a secondary KDC or to recover from a catastrophic database error.

And the **`kadmin`** command has been changed for commonality with other Kerberos implementations. Note that the kadmin command can only work on z/OS against an NDBM registry, or, on other platforms, against KDC registries that support the Kerberos V2 administration protocol.

Installing an NDBM registry

Here we briefly explain how an NDBM registry can be created. There is a chapter in *z/OS Integrated Security Services Network Authentication Service Administration*, SC24-5926, that explains in detail how to prepare the SKRBKDC task and the configuration and environment variables file to use an NDBM registry.

Note: We are assuming here that the installation is using the IBMUSER RACF user ID to log on to the z/OS UNIX shell and to interact with the NDBM KDC.

To work with an NDBM registry requires setting the SKDC_DATABASE environment variable in the envar file in the /etc/skrb/home/kdc/ directory to NDBM (as opposed to SAF when using a RACF, or equivalent, user registry. The following steps are performed to create the NDBM database itself:

1. In the z/OS UNIX shell we create the initial registry database files. In this step we use the **kdb5_ndbm** command to allocate the registry. The registry is automatically populated with two user principals: IBMUSER and IBMUSER/admin, with a password of IBMUSER. Note that the NDBM registry is protected with a master key derived from a password provided at the NDBM creation.

   We also copy the file kdc.conf from the sample directory to the actual Kerberos administrative directory.

   We make sure that IBMUSER has the KERB segment that specifies KERBNAME(IBMUSER/admin). We also make sure that IBMUSER has enough OMVS authorization to access HFS files; we use the access list of the SUPERUSER.FILESYS profile in the UNIXPRIV class for this purpose.

   Figure 8-3 shows the set of commands we issued in the z/OS UNIX shell.

```
(MVO8)-/u/yury-(OEKERN)-(131):kdb5_ndbm create
EUVF04091R Enter the KDC database master password:
<12345678>
EUVF04092R Re-enter the KDC database master password:
<12345678>
EUVF04099I KDC database created.
(MVO8)-/u/yury-(OEKERN)-(132):cd /usr/lpp/skrb/examples/
(MVO8)-/usr/lpp/skrb/examples-(OEKERN)-(133):
   cp -p /usr/lpp/skrb/examples/kdc.conf /etc/skrb/home/kdc.conf
(MVO8)-/usr/lpp/skrb/examples-(OEKERN)-(137):cd /etc/skrb/home/
```

Figure 8-3 Creation of the NDBM registry

   We do not change values in the kdc.conf file; however, each installation needs to decide about the appropriate ticket life related settings.

2. The NDBM registry uses a specific access control file, kadm5.acl, to control who has Kerberos administration privileges. Each line in the file consists of two fields: The client principal name and the privileges granted.

   We copy the example administration access control file from /usr/lpp/skrb/examples/kadm5.acl to /etc/skrb/home/kdc/kadm5.acl.

   We then use the **oedit** command to edit this file in the /etc/skrb/home/kdc/ directory. It had contained only one admin principal, IBMUSER; we add the second admin principal, YURY, with all privileges granted.

```
; Client name                  Privileges
; -----------                  ----------
  IBMUSER/admin                *
  YURY/admin                   *
```

Figure 8-4 NDBM Sample administration access control file

The administration privileges are indicated by a letter, as shown in Figure 8-5.

```
a   ADD is granted
A   ADD is denied
c   CHANGEPW is granted
C   CHANGEPW is denied
d   DELETE is granted
D   DELETE is denied
g   GET is granted (this may also be specified as i)
G   GET is denied (this may also be specified as I)
l   LIST is granted
L   LIST is denied
m   MODIFY is granted
M   MODIFY is denied
s   SETKEY is granted
S   SETKEY is denied
*   All privileges are granted
```

Figure 8-5 NDBM administration privileges

We start the SKRBKDC started procedure. Figure 8-6 on page 196 shows examples of the **kadmin** command, once the SKRBKDC server is running.

```
(MV08)-/u/yury-(IBMUSER)-(128):kadmin
EUVF06147I Authenticating as IBMUSER/admin@PSSCMOP.KDC.
EUVF02033R Enter password:

kadmin>
help
EUVF06101I Valid subcommands:
    list_principals (listprincs)
    get_principal (getprinc)
    add_principal (addprinc)
    delete_principal (delprinc)
    modify_principal (modprinc)
    rename_principal (renprinc)
    change_password (cpw)
    list_policies (listpols)
    get_policy (getpol)
    add_policy (addpol)
    modify_policy (modpol)
    delete_policy (delpol)
    get_privs (getprivs)
    add_key (ktadd)
    exit (quit)
kadmin>
addprinc -pw YURY YURY/admin
EUVF06122I Principal YURY/admin@PSSCMOP.KDC added.
kadmin>
listprincs
  kadmin/admin@PSSCMOP.KDC
  kadmin/changepw@PSSCMOP.KDC
  kadmin/history@PSSCMOP.KDC
  krbtgt/PSSCMOP.KDC@PSSCMOP.KDC
  IBMUSER@PSSCMOP.KDC
  IBMUSER/admin@PSSCMOP.KDC
  K/M@PSSCMOP.KDC
  YURY/admin@PSSCMOP.KDC
kadmin>
exit
(MV08)-/u/yury-(IBMUSER)-(129):kdb5_ndbm dump yurykdb
EUVF04102I KDC database dump file yurykdb created.
(MV08)-/u/yury-(IBMUSER)-(130): ls
cacert        env           krb5ccname   yurykdb      yurykdb.kdb  yurykdb.rdb
```

Figure 8-6 Example of use of kadmin and kdb5_ndbm

We use here a very simple subset of **kadmin** subcommands to show how the dialog works. We define a new principal, YURY in the NDBM registry, as a Kerberos admin. Then the list_principals subcommand displays the list with the two default IBMUSER principals, as well as the so-called architected principals, which must be in the registry. Furthermore, the **kdb5_ndbm** command in our example creates the transportable dump, which is really the collection of three files: yurykdb, yurykdb.kdb, and yurykdb.rdb.

IPv6 support

Another enhancement, introduced in the Network Authentication Service at z/OS 1.4, is the support for IPv6 addresses. This support translates into:

- Changes in Kerberos APIs

  – krb5_address_compare, where the address lists can contain both IPv4 and IPv6 addresses
  – APIs that generate addresses - Generated address lists can contain both IPv4 and IPv6 addresses
  – APIs that return addresses - Returned address lists can contain both IPv4 and IPv6 addresses
  – APIs that store addresses in tickets: - Address lists in tickets can contain both IPv4 and IPv6 addresses
- Changes to the GSS-API

  The Channel Bindings GSS_C_AF_INET can be replaced by GSS_C_AF_INET6.

8.2.3 Network Authentication Service enhancements in z/OS 1.6

In this section we discuss additional features, included in the z/OS 1.6 implementation of Network Authentication Service. These are:

- Using the SKRBKDC server without a local KDC
- New RACF callable service R_GenSec
- Application component trace support
- Network time offset
- Cache Functions Without a Local KDC Server running
- z990/z890 Crypto assist support

Using the SKRBKDC server without a local KDC

Up to now, the SKRBKDC started procedure was starting the Kerberos server functions such as the Authentication Server and the Ticket Granting Server. These functions require having a local KDC, be it RACF or the NDBM registry, running in the same z/OS image.

The application component trace and the sysplex credentials cache functions, which we describe later in this chapter, have been implemented via the SKRBKDC application. These functions do not have any dependency on the KDC being local. Hence the SKRBKDC server can be started by specifying that there is no local KDC running on the system. There is now a PARM keyword in the started procedure that can be either –nokdc or –kdc. You are to use -nokdc if you want to use the SKRBKDC application services (such as application component trace or sysplex credentials caches), but do not want to run a KDC on the system. Note that -kdc is the default.

RACF callable service R_GenSec for non-C programs

Standard GSS-API functions, defined in Internet RFC 2078 and RFC 2744, have been the integral part of z/OS Kerberos from day one. However, use of them is limited only to applications, written in C. There are Kerberos-enabled applications that are not written in C; furthermore, setting up language environment just for Kerberos functions introduced an undesirable overhead.

To allow non-C applications to establish a session and exchange messages between the context initiator and the context acceptor, a new RACF callable service, R_GenSec(IRRSGS00), provides for fourteen subfunction codes, which are quite close to their GSS-API counterparts. Here is the table, in Figure 8-7 on page 198 which shows the codes and their corresponding to GSS-API functions.

1	Initiate a GSS-API security context	*gss_init_sec_context*
2	Continue initiation of a GSS-API security context	see conditions in usage of *gss_init_sec_context*
3	Accept a GSS-API security context	*gss_accept_sec_context*
4	Delete a GSS-API security context	*gss_delete_sec_context*
5	Release a GSS-API credential	*gss_release_cred*
6	Get the MIC for a message	*gss_get_mic*
7	Verify the MIC for a message	*gss_verify_mic*
8	Wrap a message	*gss_wrap*
9	Unwrap a message	*gss_unwrap*
10	Export a GSS-API security context	*gss_export_sec_context*
11	Import a GSS-API security context	*gss_import_sec_context*
12	Export a GSS-API credential	*gss_export_cred*
13	Import a GSS-API credential	*gss_import_cred*
14	Acquire a GSS-API initiator credential	*gss_acquire_cred*

Figure 8-7 R_GenSec callable services

Since the new service is dealing with GSS-API context tokens, callers, not running in system key or supervisor state, need to be authorized for the use of R_GenSec. It is done via the existing resource IRR.RTICKETSERV in the FACILITY class for function code 1. Additionally, for the same unauthorized callers, subfunction code 2 is protected by resource IRR.GSSERV in the FACILITY class. The application server must be running with a RACF user or group ID that has at least READ authority to the resources. If the class is inactive, or the resource is not defined, only servers running with a system key or in supervisor state may use the R_GenSec service.

Application component trace support

Kerberos component trace is not a new option, it has existed in pre-z/OS 1.6 releases of Kerberos. However, the new release allows starting and stopping, and then starting again the component trace, using the regular TRACE CT command on the system console, issued for the SKRBKDC component. This may be done with or without the application running.

If an installation needs to gather the trace on a system, which is not used for the KDC, then SKRBKDC is to be started with PARMS='-nokdc'.

In order to collect a trace in an external data set rather than in the address space, the SKRBWTR external writer procedure is provided in the EUVF.SEUVFSAM sample library. Our procedure had a slightly modified TRCOUT01 DD statement, with respect to the sample, as shown in Figure 8-8.

```
//SKRBWTR  PROC
//*-----------------------------------------------------------------*//
//*  Component trace external writer for the Kerberos security      *//
//*  server.  Up to 16 trace datasets (TRCOUT01 - TRCOUT16) may     *//
//*  be specified.                                                  *//
//*-----------------------------------------------------------------*//
//IEFPROC  EXEC  PGM=ITTTRCWR,REGION=32M
//TRCOUT01  DD   DSN=SKRBKDC.TRACE,DISP=(NEW,CATLG),
//    SPACE=(CYL,(2,1)),UNIT=SYSALLDA
```

Figure 8-8 Sample SKRBWTR started procedure

We recommend that you do as we did ourselves—associate the SKRBWTR started procedure with the same user ID as assigned to the SKRBKDC address space. In our case it is SKRBKDC. We then started the writer via the TRACE CT,WTRSTART=SKRBWTR system command, and activated the Kerberos application component trace via the TRACE CT,ON,COMP=SKRBKDC system component. Here is the log of these commands, in Figure 8-9, up to the point of the full stop of the trace.

```
- starting the application component trace

TRACE CT,ON,COMP=SKRBKDC
*02 ITT006A SPECIFY OPERAND(S) FOR TRACE CT COMMAND.
 R 2,JOBNAME=(IBMUSER),OPTIONS=(*.1),WTR=SKRBWTR,END
 IEE600I REPLY TO 02 IS;JOBNAME=(IBMUSER),OPTIONS=(*.1),WTR=SKRBWTR
 EUVF04150I Component trace started.
 ITT038I ALL OF THE TRANSACTIONS REQUESTED VIA THE TRACE CT COMMAND WERE
 SUCCESSFULLY EXECUTED.
IEE839I ST=(ON,0256K,01024K) AS=ON  BR=OFF EX=ON  MT=(ON,064K) 632
        ISSUE DISPLAY TRACE CMD FOR SYSTEM AND COMPONENT TRACE STATUS
        ISSUE DISPLAY TRACE,TT CMD FOR TRANSACTION TRACE STATUS
EUVF04152I Component trace started for IBMUSER.

- using kadmin in the IBMUSER session and dumping the KDC registry (see kadmin example
in Figure 8-6 on page 196)

- stopping the application component trace

TRACE CT,OFF,COMP=SKRBKDC
EUVF04151I Component trace ended.
ITT038I ALL OF THE TRANSACTIONS REQUESTED VIA THE TRACE CT COMMAND WERE
SUCCESSFULLY EXECUTED.
IEE839I ST=(ON,0256K,01024K) AS=ON  BR=OFF EX=ON  MT=(ON,064K) 640
        ISSUE DISPLAY TRACE CMD FOR SYSTEM AND COMPONENT TRACE STATUS
        ISSUE DISPLAY TRACE,TT CMD FOR TRANSACTION TRACE STATUS
TRACE CT,WTRSTOP=SKRBWTR
ITT038I ALL OF THE TRANSACTIONS REQUESTED VIA THE TRACE CT COMMAND WERE
SUCCESSFULLY EXECUTED.
IEE839I ST=(ON,0256K,01024K) AS=ON  BR=OFF EX=ON  MT=(ON,064K) 644
        ISSUE DISPLAY TRACE CMD FOR SYSTEM AND COMPONENT TRACE STATUS
        ISSUE DISPLAY TRACE,TT CMD FOR TRANSACTION TRACE STATUS
AHL904I THE FOLLOWING TRACE DATASETS CONTAIN TRACE DATA : 643
          SKRBKDC.TRACE
ITT111I CTRACE WRITER SKRBWTR TERMINATED BECAUSE OF A WTRSTOP REQUEST
IEF196I IEF142I SKRBWTR SKRBWTR - STEP WAS EXECUTED - COND CODE 0000
```

Figure 8-9 Example of starting and stopping the Kerberos application component trace

The produced data set may be processed in IPCS using the IPCS CTRACE COMP(SKRBKDC) command. We did this in the IPCS dialog from the Option 6 “Enter subcommand, CLIST or REXX exec” panel we issue CTRACE COMP(SKRBKDC) FULL. Here is an excerpt from the produced output in Figure 8-10 on page 200, showing the creation of the new principal YURY in the Kerberos database, using the kadm5 API.

```
 COMPONENT TRACE FULL FORMAT
  SYSNAME(MV08)
  COMP(SKRBKDC)
  **** 09/06/2004

  SYSNAME   MNEMONIC  ENTRY ID    TIME STAMP     DESCRIPTION
  -------   --------  --------  ---------------  -------------

  MV08      TRACE     00000001  14:55:02.405301  KRB_GENERAL

    Job IBMUSER  PID 02000055 TID 00000000 LVL 1 krb5_svc_begin_ctrace
    Version 3, Release 16, Service level 0000000
. . . . . . . . .
. . . . . . . . .
. . . . . . . . .
MV08      TRACE     00000000  14:58:28.322190  KRB_API

    Job IBMUSER  PID 02000055 TID 00000000 LVL 1 krb5_unparse_name
    <-- krb5_unparse_name(1): Status 0x0, Name YURY/admin@PSSCMOP.KDC

  MV08      TRACE     00000000  14:58:28.322244  KRB_API

    Job IBMUSER  PID 02000055 TID 00000000 LVL 1 kadm5_create_principal
    --> kadm5_create_principal(): Handle 19fddb60, Mask 1, Principal YURY/admin
```

Figure 8-10 Output produced by CTRACE COMP(SKRBKDC) FULL command in IPCS

Network time offset

Some configurations may require setting up KDCs running in different time zones while still connected to the same network. There is then a requirement that the SKRBKDC server be able to time stamp tickets with a time different than the time of the system it is running on, so that all KDCs are using the same network time.

A new clock_offset parameter has been introduced in the [libdefaults] section of the /etc/skrb/krb5.conf file. The specified offset is added to the system clock to obtain the network time. The default is 0.

The use of the clock_offset parameter value depends on the existing kdc_timesync parameter, that is, if kdc_timesync is 0 (the default) then Kerberos uses the clock_offset value to compute the network time. If, however, kdc_timesync is 1, then the clock_offset is ignored. An example of clock_offset setup is shown in Figure 8-11.

```
[libdefaults]
    clock_offset = 300
```

Figure 8-11 Example of clock_offset

This value is in minutes, and added to the system's clock time. In the provided example the assumption is that the system clock is set at GMT and the network time is GMT +5 hours.

Note: The clock_offset allows you to specify an offset from the network time. That is, the offset is added to the z/OS clock to get the current network time. The kdc_timesync allows a client to synchronize its clock with the KDC. The difference is the first method assumes z/OS does not have the same time base as the network clients while the second option allows network clients to set their clocks based on the KDC time (that is, they share the same time base).

Cache functions without a local KDC server running

Normally when the KDC is not local to the current system, the credentials have to be cached in an HFS file. Each member in a sysplex would have to cache their credentials locally. Now these systems can use a sysplex cache.

With this enhancement, the SKRBKDC address space, representing the KDC, joins the sysplex group EUVFSKRB, which may be joined by other systems in the same sysplex, not running their own KDC. Using sysplex services, they may have access to the credentials, cached in an HFS file on the KDC system, via the EUVFSKRB sysplex group.

There is nothing to be changed in the Kerberos customization process. However, you need to account for yet another sysplex group with as many members as the number of systems in the sysplex when formatting a new sysplex couple data set.

z990/z890 Crypto assist support

Kerberos has been enhanced to benefit from the new cryptographic facility in the zSeries z990/z890 models, known as CPACF (CP Assist for Cryptographic Functions). The CPACF is the standard hardware cryptographic facility that is provided in place of the Cryptographic Coprocessor Feature (CCF), which is available in the other models. Kerberos now supports the use of CPACF, as well as CCF. It recognizes the installed hardware feature, and uses it appropriately. This support is absolutely transparent to installations, and does not require any actions during the Kerberos setup.

8.3 GSS-API and krb5_ API test programs

Test programs, written in C, are delivered in source code with the z/OS Network Authentication Service. They can be used to check out a Kerberos setup by exercising the Kerberos APIs, either the GSS-API or the krb5_ API. They can be found in /usr/lpp/skrb/examples/gssapi_test/ and in /usr/lpp/skrb/examples/krbmsg_test/. In this chapter we give an overview of the GSS-API test setup and operation.

Additional information about what these C programs do and how to compile them are given in the README file in the same directories.

The GGS-API test

The gssapi_test application is intended to test the following GSS-API calls:

- gss_init_sec_context()
- gss_accept_sec_context()
- gss_get_mic()
- gss_verify_mic()
- gss_wrap()
- gss_unwrap()
- gss_delete_sec_context()
- gss_process_context_token()

These functions are being used in a configuration with a Kerberos client, an intermediate server, and an end-server, using a Kerberos forwardable ticket provided by the client. A schematic view of the interactions is given in Figure 8-12.

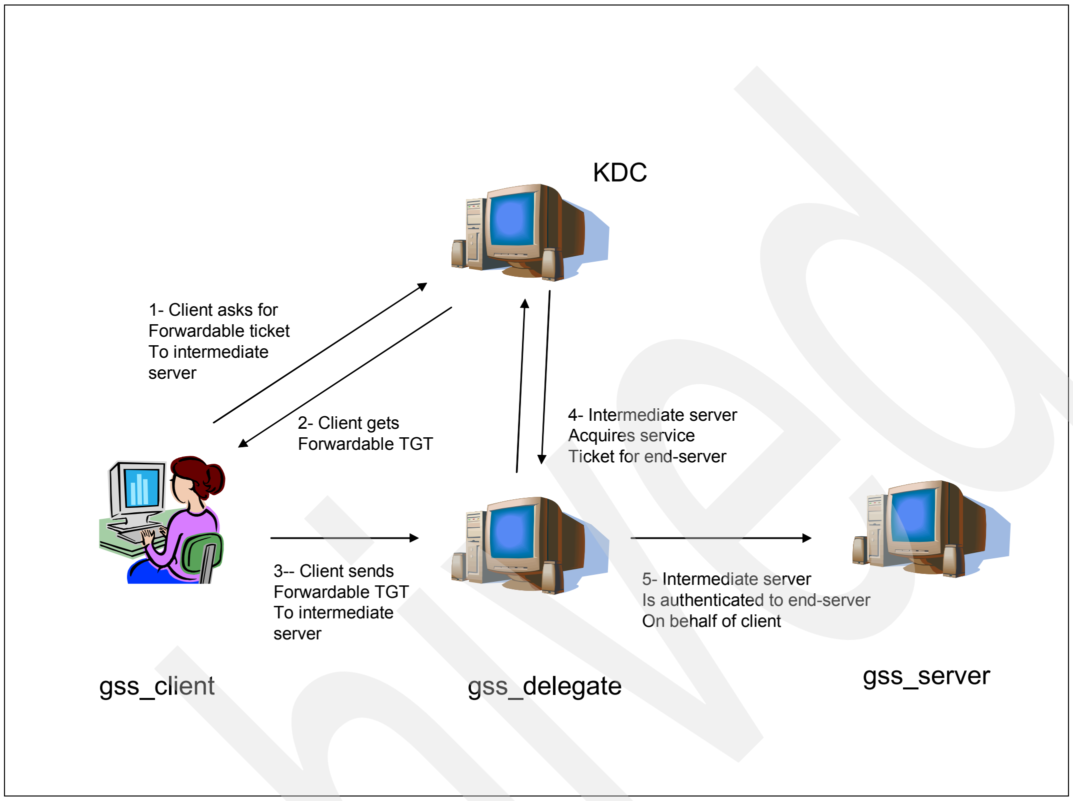

KDC
1- Client asks for Forwardable ticket To intermediate server
2- Client gets Forwardable TGT
4- Intermediate server Acquires service Ticket for end-server
3-- Client sends Forwardable TGT To intermediate server
5- Intermediate server Is authenticated to end-server On behalf of client
gss_client
gss_delegate
gss_server

Figure 8-12 gssapi test interactions

8.3.1 Setup

We are using a single z/OS image hosting all entities:

- The Kerberos KDC, which is the RACF instance in our z/OS image
- The Kerberos client, actually the C program called gss_client
- The intermediate server (gss_delegate)
- The end-server (gss_server)

Clients and servers are given a RACF user ID along with a Kerberos principal name, as explained further below.

8.3.2 The RACF KDC

The SKRBKDC server has been configured as shown in Figure 8-13 on page 203, and the RACF realm has been defined as:

```
ralt realm kerbdflt kerb(kerbname('PSSCMOP.KDC'))
```

The TCPIP host name in the TCP/IP configuration files is MVO8, with a domain origin of PSSCMOP.IBM.COM. As we are not using a DNS, we have defined the domain and realm in the /etc/skrb/krb5.conf, as shown in Figure 8-13 on page 203, and we have added the following statement in the /etc/hosts file:

```
10.11.12.13 MVO8 MVO8.PSSCMOP.IBM.COM
```

```
[libdefaults]

default_realm = PSSCMOP.KDC
kdc_default_options = 0x40000010
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]

PSSCMOP.KDC = {
    kdc = mvo8.psscmop.ibm.com:88
    kpasswd_server = mvo8.psscmop.ibm.com:464
    admin_server = mvo8.psscmop.ibm.com:749
}


[domain_realm]
.psscmop.ibm.com = PSSCMOP.KDC
```

Figure 8-13 SKRBKDC test configuration

8.3.3 The client

We have created a RACF USER profile for the client with a user ID of KERBCLI and a KERBNAME of 'KERBCLI/MVO8'.

We have compiled the gss_client.c program as indicated in the README file with a name of gss_client, and we have inserted in the OMVS shell profile of the client, as in the statement shown below:

```
export KRB5_SERVER_KEYTAB=1
```

Actually, the intermediate server and end-server users will also have the same statement in their .profile file.

8.3.4 The intermediate server

The intermediate server has been given the RACF user ID KERBDLG, with a KERBNAME of 'test_delegate/MVO8', as required in the README file. The test program has been compiled as gss_delegate.

8.3.5 The end-server

The end-server has been given the RACF user ID KERBSRV with a KERBNAME of test_server/mvo8.The test program has been compiled as gss_server.

8.3.6 Running the test

Running the test is done by getting the three users being logged on the system and going, for each of them, into the z/OS UNIX shell as explained below.

We recommend entering the z/OS UNIX shell with the OMVS ESCAPE('@') command, for it will allow you to use the @C sequence to stop, if required, the shell foreground tasks of gss_server and gss_delegate programs via the SIGINT signal.

The first program to be started is gss_server; therefore, we log on to TSO/E as KERBSRV and then enter OMVS ESCAPE('@'). We then type in the name of the compiled C program, as shown in Figure 8-14. The end-server starts and indicates which port number it is listening at. This is a random port—20014 in this example.

```
(MVO8)-/u/yury-(OEKERN)-(20):gss_server
gss_server: GSS server starting
gss_server: Local host is mvo8
gss_server: Acceptor credential mechanism 1: KRBV5_DES_RFC
gss_server: Acceptor credential mechanism 2: KRBV5_DES_BETA
gss_server: Listening for requests on port 20014
```

Figure 8-14 Starting the gss_server program

Next we log on to TSO/E as the user KERBDLG, go into the z/OS UNIX shell, and start the gss_delegate program. Note that we are required to enter the domain names of the server and the port it is listening at (20014 in our case, as indicated by the gss_server program). The intermediate server indicates which port it is listening at for a client's request, as shown in Figure 8-15.

```
(MVO8)-/u/kerbdlg-(OEKERN)-(17):gss_delegate mvo8.psscmop.ibm.com 20014
gss_delegate: GSS delegate starting
gss_delegate: Local host is mvo8
gss_delegate: Server host is mvo8
gss_delegate: Acceptor credential mechanism 1: KRBV5_DES_RFC
gss_delegate: Acceptor credential mechanism 2: KRBV5_DES_BETA
gss_delegate: Listening for requests on port 20015
```

Figure 8-15 Starting the gss_delegate program

Finally, we log on as KERBCLI and go into the z/OS UNIX shell. Before starting the client program we need to acquire from the KDC an authentication ticket. This is done by issuing the **kinit** command, as shown Figure 8-16 on page 205. Note the parameters of the kinit command: -s to indicate that we want a ticket for the logged on user (that is, we are not prompted again for a password in order to get the ticket), and -f to ask for a forwardable ticket. The **kinit** command is followed by a **klist** command to verify that we now actually have a credential in the client credential cache.

The next step is to start the gss_client program using the port number provided by the gss_delegate program upon its startup (here, 20015). The client, intermediate server, and end-server then interact and issue the messages shown in Figure 8-16 on page 205.

```
   End Server shell messages:
gss_server: Connection received from 9.100.193.71[20140]
gss_server: GSS context 1 established
    Context lifetime=35960 seconds, Context flags=000001be
    Context mechanism=KRBV5_DES_RFC
    Context initiator=kerbcli/mvo8@PSSCMOP.KDC
    Context initiator=kerbcli/mvo8@PSSCMOP.KDC
    Client says: Greetings from a faithful client who seeks enlightment as to th
e ultimate meaning of digital communications
    Signature QOP=0002, Seal QOP=0102
    Client greeting is encrypted
gss_server: Connection closed with 9.100.193.71[20140]

   Intermediate Server shell messages:

gss_delegate: Connection received from 9.100.193.71Ý20139¨
gss_delegate: GSS context 1 established
    Context lifetime=35960 seconds, Context flags=000001b1
    Context mechanism=KRBV5_DES_RFC
    Mutual authentication not required
    Context initiator=kerbcli/mvo8@PSSCMOP.KDC
gss_delegate: Signature QOP=0002, Seal QOP=0102
gss_delegate: Connection closed with 9.100.193.71Ý20139¨

   Client shell messages:

(MVO8)-/u/kerbcli-(KERBCLI)-(2):kinit -s -f
(MVO8)-/u/kerbcli-(KERBCLI)-(3):
(MVO8)-/u/kerbcli-(KERBCLI)-(3):klist
 Ticket cache: FILE:/var/skrb/creds/krbcred_1369b8b0
 Default principal: kerbcli/mvo8@PSSCMOP.KDC

Server: krbtgt/PSSCMOP.KDC@PSSCMOP.KDC
  Valid 2004/09/02-06:03:23 to 2004/09/02-16:03:23

(MVO8)-/u/kerbcli-(KERBCLI)-(5):gss_client mvo8.psscmop.ibm.com 20015
gss_client: GSS client starting
gss_client: Initiator credential mechanism 1: KRBV5_DES_RFC
gss_client: Initiator credential mechanism 2: KRBV5_DES_BETA
gss_client: KRBV5_DES_RFC mechanism, Mutual auth FALSE, Seq check FALSE
gss_client: Signature QOP=0002
gss_client: Wrapped message QOP=0102
gss_client: Server reply: Server greetings at Thu Sep  2 05:41:39 2004
gss_client: KRBV5_DES_RFC mechanism, Mutual auth FALSE, Seq check TRUE
gss_client: Signature QOP=0002
gss_client: Wrapped message QOP=0102
gss_client: Server reply: Server greetings at Thu Sep  2 05:41:39 2004
gss_client: KRBV5_DES_RFC mechanism, Mutual auth TRUE, Seq check FALSE
gss_client: Signature QOP=0002
gss_client: Wrapped message QOP=0102
gss_client: Server reply: Server greetings at Thu Sep  2 05:41:39 2004
gss_client: KRBV5_DES_RFC mechanism, Mutual auth TRUE, Seq check TRUE
gss_client: Signature QOP=0002
gss_client: Wrapped message QOP=0102
gss_client: Server reply: Server greetings at Thu Sep  2 05:41:39 2004
gss_client: KRBV5_DES_BETA mechanism, Mutual auth FALSE, Seq check FALSE
gss_client: GSS client ending
```

Figure 8-16 Starting the gss_client program, and messages issued by the three programs

As soon as the client program issues its `GSS client ending` message, we can type a **‘**`@C`**’** string in the server and intermediate server shell sessions to stop the applications.

9

z/OS System SSL

This chapter provides the status of the System SSL component of z/OS as of z/OS 1.6. It also reviews the System SSL improvements since its initial release at OS/390 2.7.

System SSL has already been described in the redbooks *OS/390 Security Server 1999 Updates Installation and Implementation Guide*, SG24-5629, and *OS/390 Security Server 1999 Updates Technical Presentation Guide*, SG24-5627.

The reference book for System SSL is *z/OS Cryptographic Services System Secure Sockets Layer Programming*, SC24-5901.

9.1 SSL and TLS reminder

Secure Socket Layer is a client-server communication protocol developed by Netscape for securing communications that use TCP/IP sockets. It uses asymmetric (public key) and symmetric key cryptography:

- For server authentication
- To provide data privacy and integrity
- For optional client authentication via digital certificate

Note that the SSL protocol is executed at the application level and therefore an application has to be designed to support SSL in order to benefit from the SSL protection. It is also interesting to notice that, although SSL is very well known as a way to secure http communications, any kind of TCP/IP socket communication is a candidate for SSL protection, with proper code support. For instance, the z/OS TN3270 server and the FTP and LDAP servers and clients are SSL enabled. This indirectly adds to the SSL protocol popularity, as its implementation does not require any specific TCP/IP stack, network setup, or administration (the SSL communication is just TCP protocol as far as the IP packets are concerned). This is illustrated in Figure 9-1, where the SSL TCP packet is shown to transport three kinds of information built by the application SSL code:

- SSL flags indicating the type of information transported. That is the type of *SSL message*.
- A cryptographic integrity checking (typically using the MD5 or SHA-1 algorithm).
- The encrypted data (typically using the RC2, RC4, DES, or Triple DES algorithms). Note also that SSL is also segmenting the data to be transmitted.

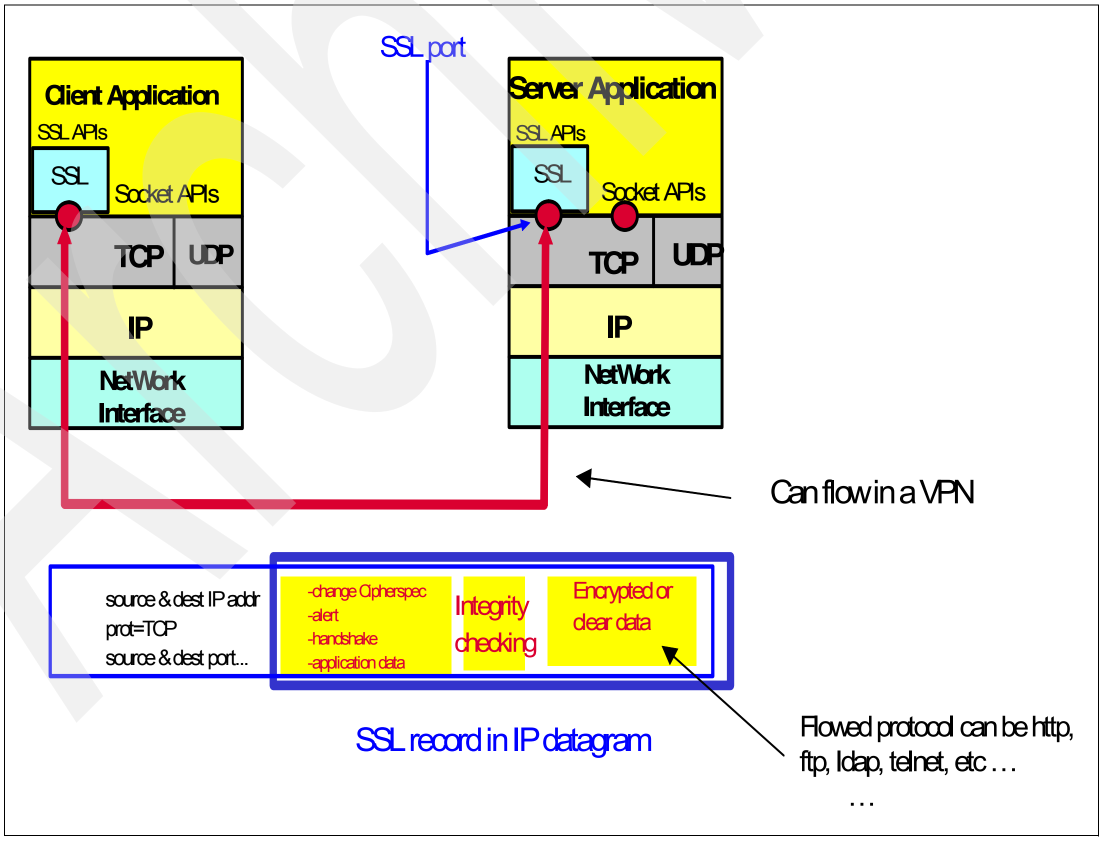


Figure 9-1 The SSL TCP packets

However, SSL is also known to be a very demanding protocol in terms of computing resources in the session initialization phase known as the *handshake*. These resources are needed to support the heavy calculation required by the asymmetric (that is, public key) cryptography that the handshake uses. Hence the advantage of having hardware cryptographic facilities these calculations can be off-loaded to.

The handshake is followed by a data transfer phase, also called the SSL *record protocol*, which uses symmetric cryptography. Symmetric cryptography is by far less computing intensive than asymmetric cryptography, and it might be quite acceptable, in environments with moderate SSL utilization, to perform this encryption with software only.

9.1.1 The SSL protocol interactions sequence

A schematic view of the SSL protocol is given in Figure 9-2 on page 210. The sequence of events is:

1. The client connects to the server indicating that it wants to perform SSL (`client hello` SSL message). Note that the client hello message also conveys a sessionID. The use of the sessionID is explained later.
2. The server agrees to carry on an SSL conversation (“server hello”). The handshake begins.
3. The client and server agree on a common symmetric algorithm to be used for the data transfer that follows the handshake. The server imposes the list of possible algorithms and its order of preference in this list, which is compared to the client’s own list.
4. The server provides its public key in its certificate to the client—this is also called *server authentication*, and is a mandatory step of the SSL handshake.
5. The client verifies the integrity of the server’s certificate, using the certificate of the Certification Authority that signed the server’s certificate. Depending on the client design, if this certificate is not available due to an incomplete setup of the client’s system or because the server’s Certification Authority is not known from the end user, the client may ask the end user to agree on pursuing the communication or to abort it.
6. The client now has the server’s public key and uses it to encrypt a random number, which is then sent to the server. The server retrieves the value of this random number using its private key. Note that it is the decryption of the secret random number using the server’s private key that costs so much computing resource during the SSL handshake. This operation can be avoided, as explained below, by re-using the so-called session ID.
7. At this point only the client and server know this secret random number (the pre master secret), which can then be used to generate the keys to encrypt and decrypt the data, using the symmetric algorithm previously selected. The client and the server then close the handshake phase and enter the data transfer phase (also called SSL record protocol).

A couple of remarks here:

- As an option the client can be asked to provide a certificate as means of authentication. This also implies that the client afterwards in the handshake sequence presents a digital signature that demonstrates that it also has the private key that goes with this very certificate.
- Every time a new random number is exchanged with a server, the server tags this pre-master secret with an SSL sessionID, which is also transmitted to the client. This session ID can be thought of as an index to retrieve a previously used pre-master secret. The sessionID is not a secret, but is of no use for whoever does not know the value of the corresponding pre-master secret.

- The pre-master secret value is actually used to derive the values of several different symmetric keys on each side: Keys for data encryption and decryption in each direction and keys for data integrity checking.

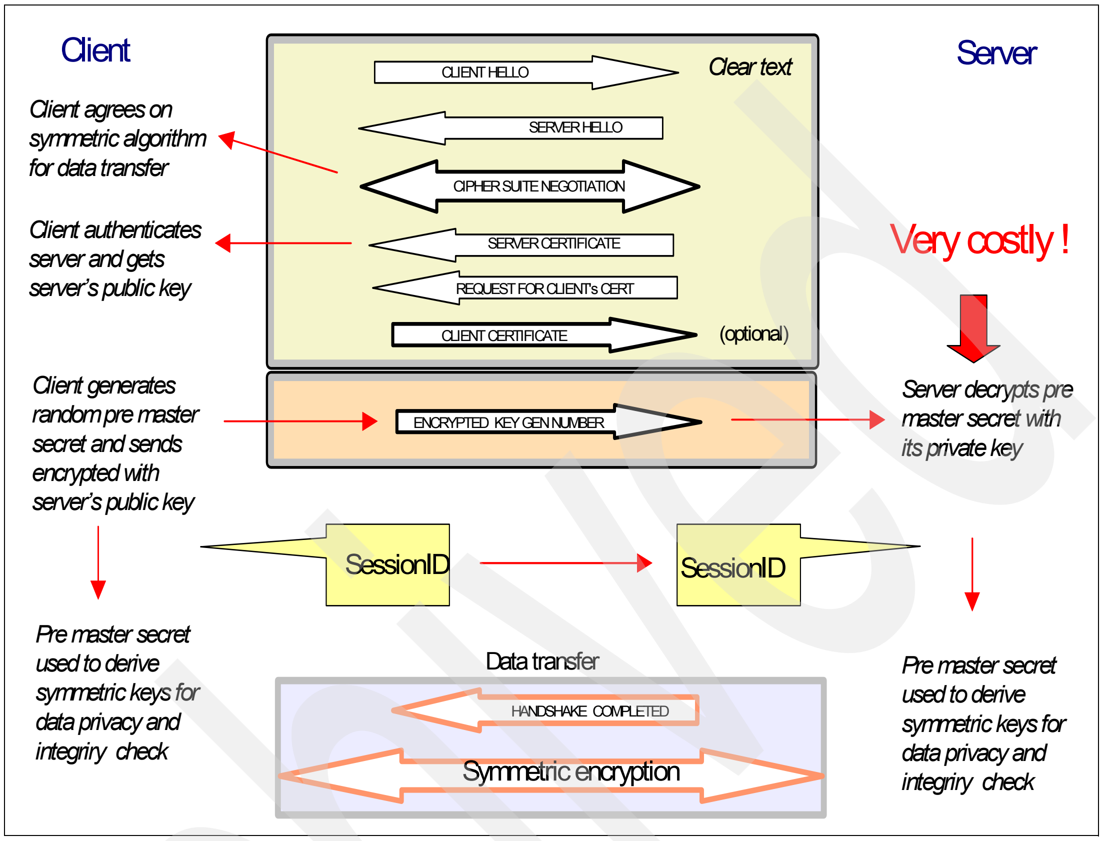

Client
Server
Clear text
CLIENT HELLO
SERVER HELLO
CIPHER SUITE NEGOTIATION
SERVER CERTIFICATE
REQUEST FOR CLIENT's CERT
CLIENT CERTIFICATE
(optional)
Client agrees on symmetric algorithm for data transfer
Client authenticates server and gets server's public key
Very costly !
Client generates random pre master secret and sends encrypted with server's public key
ENCRYPTED KEY GEN NUMBER
Server decrypts pre master secret with its private key
SessionID
SessionID
Pre master secret used to derive symmetric keys for data privacy and integriry check
Data transfer
HANDSHAKE COMPLETED
Symmetric encryption
Pre master secret used to derive symmetric keys for data privacy and integriry check

Figure 9-2 The SSL protocol

9.1.2 The sessionID re-use

It is important to understand when an SSL handshake is actually to occur and how one can reduce the amount of these costly handshakes. A handshake has to occur every time a TCP connection is initiated by the client, which means that SSL protected protocols that are keeping the connection open for a long time, such as TN3270 does, are executing handshakes very seldom. On the other hand, protocols that are closing the connection frequently, as http should do after each html page, are then very costly in handshakes. The use of persistent sessions with http 1.1 is then reducing the number of handshakes.

The SSL protocol has a built-in mechanism to reduce the number of handshakes: The sessionID re-use. Actually, the SSL server can be set up to remember the pre-master secret previously established with the client and the corresponding sessionID. When a client re-establishes a TCP connection to resume an SSL conversation with a server, it actually presents the previous sessionID that was established with this server in the client hello message. If the server remembers the pre-master secret linked to this sessionID, it then signals the client that a new handshake is not needed; they will re-use the previous pre-master secret value. They then directly enter the data transfer phase.

Remarks:

- The server is set up to remember the sessionID with the SSLtimeOut parameter, which tells for how long the server keeps the previous pre-master secret in its memory. The duration of the SSLtimeOut, commonly in the order of several minutes or hours, is an installation trade-off between security and performance, as the longer the timeout the longer the time left to an attacker to discover the value of the symmetric keys.

- Even if the pre-master secret value is reused, the derived keys are different every time from the keys actually used in the previous session, which reduces the exposure to a key discovery attack.

9.1.3 SSL and TLS

The first version of SSL, in 1994 was never released by Netscape. Only Version 2 (1995) and Version 3 (1996) were made available for public use. SSL V3 fixed security problems found in SSL V2, offered more options for data privacy and integrity checking, and allowed a client to authenticate using public key cryptography, the so-called SSL client authentication. SSL does not exist any more as such; it is now replaced by the Transport Layer Security (TLS) protocol, which is the SSL V3 protocol revisited and standardized by the Internet Engineering Task Force (IETF) in RFC 2246. TLS is not interoperable with SSL; however, TLS provides the capability to revert to SSL when communicating with a non-TLS-capable partner. Note that TLS is sometimes referred to as SSL V3.1.

9.1.4 SSL client authentication

As an option with the SSL V3 or the TLS protocols, the SSL/TLS server can optionally request a client certificate, which could eventually be used for authentication purposes if the intended URL is protected. If the client provides a certificate, it also provides a digital signature demonstrating that it also possesses the private key that corresponds to the certificate.

Note that the SSL/TLS server then needs to have the client's Certification Authority public key (that is, the Certification Authority certificate) to verify the client certificate.

> **Note:** SSL client authentication does not go farther than just demonstrating that the client does have the corresponding private key, and therefore the authenticated identity is the certificate's identity. Meaning here that RACF is not involved at all in SSL client authentication. But it can become involved later when mapping the authenticated certificate to a local z/OS user ID.

The SSL/TLS server can also perform, as an option, a checking of the client certificate revocation status by getting a Certificate Revocation List (CRL) from an LDAP directory entry managed by the client's Certification Authority.

9.2 z/OS System SSL

System SSL has been introduced in OS/390 2.7 as a a set of libraries that are called by APIs provided to C/C ++ applications protecting their TCP/IP sockets communications with SSL. Hence these applications only have to properly call the System SSL API, as opposed to having been designed with complete SSL support embedded. These applications can act either as an SSL client or an SSL server, and should call the API functions accordingly.

System SSL also comprises a UNIX utility to create and manage key pairs and certificates: The gskkyman utility.

To follow export regulations, System SSL was initially delivered with a base FMID, which could be enhanced by one out of two other FMIDs that are allowing longer cryptographic keys to be used. As it stands today, there is only one optional FMID to be ordered to replace the base FMID when the length of the symmetric key used for the data transfer is to be greater than 56 bits (which is the case with Triple DES or 128-bit RC4). The FMID to order is the "Cryptographic Services Security Level 3 FMID.

There are roughly two families of APIs in the z/OS System SSL:

- The API to be used to invoke and manage the execution of an SSL-protected communication. The API is available to SSL client and SSL server applications.
- The API that an application can use to automate the certificate management functions that are usually done in an interactive way by an end user invoking the gskkyman utility. Note that this API also provides PKCS#7 functions. These latter functions are exploited, under the covers, by RACF when it performs the new Password Enveloping function at z/OS 1.5 and later.

System SSL has also embedded diagnostic and tracing facilities.

9.2.1 Packaging at z/OS 1.6

At z/OS 1.6 System SSL is delivered in the *pdsname*.SIEALNKE data set. It was the *pdsname*.SGSKLOAD prior to z/OS 1.6. The base SSL FMID is HCPT360 and the optional ID Cryptographic Services Security Level 3 FMID. JCPT361 must be installed when intending to use symmetric keys longer than 56 bits.

9.2.2 Exploiters of System SSL

z/OS System SSL is exploited today by the following z/OS servers and clients:

- http server
- WebSphere Application Server
- CICS/TS, TN3270
- IMS™ Connect
- FTP server and client
- LDAP server and client
- WebSphere MQ (Tivoli AMBI)

9.2.3 System SSL history

In this section we review System SSL history.

OS/390 2.7

This is the initial implementation, with support for SSL V2 and V3, with optional client authentication. The SSL handshake uses the RSA algorithm both for certificate signature and pre-master secret protection. The keys and certificates used by the SSL libraries must reside in HFS files only and are managed by the gskkyman utility. The gskkyman utility is further described at 9.5, "Managing keys and certificates with gskkyman" on page 222.

OS/390 2.8

These release introduces the capability to use keys and certificates stored in the RACF database and connected to RACF keyrings. The use of RACF keyrings is described at 9.6, "Managing keys and certificates with RACDCERT command" on page 229.

OS/390 2.10

A limited Java interface is provided, which is to be eventually dropped at z/OS 1.6. The IBM strategic support of Java SSL is currently JSSE.

This release introduces the support of private keys stored in the ICSF PKDS data set.

z/OS 1.2

TLS V1.0 is supported. A single process can use multiple SSL environments (that is, sets of parameters) selected, for instance, on the basis of which client is currently being served by the process.

z/OS 1.4

System SSL has gone under a major rewrite for performance improvement. AES is supported with keys of 128 and 256 bits, and therefore requires that the Security Level 3 FMID be installed.

Other miscellaneous support includes:

- Digital Signature Standard (DSS) certificate support - System SSL supports client certificates signed with the Digital Signature standard algorithm (DSA).
- IPV6 addressing - System SSL is sensitive to the format of IP addresses when calling, for instance, TCP/IP functions such as getpeername, or when building the sessionID in the in-storage cache.
- Session ID caching across sysplex - This function is intended to allow members of a sysplex to share the stored session IDs. This function is further explained in “A word on SessionID re-use - Sysplex SessionID caching” on page 216.
- CRL caching - The CRL obtained from an LDAP directory can be cached for a specified amount of time in order to avoid getting again to the directory for another access to the same CRL.
- RACF keyring sharing - RACF keyrings can be shared between SSL-enabled applications, specifying who is the owner of the keyring and the name of the keyring; typically this is indicated to the application by a parameter like *owner_name/keyring_name*. The using user ID, usually the application user ID, must be permitted with at least UPDATE access to the IRR.DIGTCERT.LISTRING RACF profile in the FACILITY class. Note that private keys can never be read by somebody other than the private key’s owner, even if the keyring the key is connected to is accessible to several users.

z/OS 1.6

For z/OS 1.6:

- Support of the Diffie-Hellman (fixed and ephemeral) algorithm for SSL/TLS handshakes and certificates, in addition to the RSA algorithm.

  Diffie Hellman is a public key cryptography protocol used to dynamically establish a shared secret symmetric key between two partners. The TLS specification requires Diffie-Hellman to be one of the possible protocols that can be used during the handshake to establish the pre-master secret.

Diffie-Hellman use notes:

- Support has been added to the SSL handshake for fixed and ephemeral Diffie-Hellman key exchanges. These are new cipher values to be specified for the gsk_secure_soc_init() and gsk_secure_socket_init() System SSL functions.
- Fixed Diffie-Hellman uses the same key pair for each secure connection being established. DH parameters and keys are obtained from the server's certificate.
- Ephemeral Diffie-Hellman uses different key pairs for each secure connection established. Pre-generated Diffie-Hellman and temporary public/private keys are used.
- Anonymous Diffie-Hellman is not supported by z/OS System SSL.
- The server side of the connection decides which cipher is used.
- If the client uses a Diffie-Hellman certificate, it *must* use the same group and generator parameters as the server.

- Enhanced server certificate validation - This an option for an SSL client application to accept the server certificate only if the subject's name CN in the certificate is the same as the name used by the client to call the server. Or, alternatively, if the same name appears in the server's certificate as a DNS alternate name extension.
- z990 Cryptographic Coprocessors support - System SSL exploits the new z990 CPACF (CP Assist for Cryptographic functions) for symmetric encryption and the PCICA, PCIXCC, or Crypto Express2 coprocessors for the handshake. This support is available on previous z/OS release via PTF.
- System SSL is enabled for 64-bit addressing – If the hardware crypto is to be invoked then the ICSF version must also support the 64-bit addressing. This support in ICSF is provided beginning with FMID HCR7720.
- The Java support introduced at OS/390 2.10 is removed – The IBM recommendation is to use IBM JSSE instead.

9.3 System SSL principles of operations

Using the System SSL API, an application can:

- Provide environment parameters at the UNIX process level: That is, parameters that specify where the key pair and certificates to use are located (which HFS key database or RACF keyring to use), the LDAP URL to be used for CRL checking, etc.
- Establish parameters at a lower level for each secure connection established and served by a thread in the process, such as the file descriptor of the connection, the label of the server certificate to be presented for this connection, etc.
- Trigger an SSL handshake, or transparently reuse a sessionID when resuming the session.
- Retrieve the received data after they have been decrypted or provide data to be encrypted then to be sent.

Note: System SSL transparently invokes the hardware crypto, if available on the system, to perform the RSA computation required by the handshake and for the data encryption if DES or T-DES have been negotiated between the SSL client and server. The exploitation of the cryptographic coprocessors is further explained in 9.4.2, “Invocation of the hardware cryptographic coprocessors” on page 219.

Figure 9-3 on page 216 is a schematic view of the z/OS System SSL implementation. The SSL-enabled server, on the right side, invokes the System SSL API in the center of the picture once it has established the TCP connection with the client; that is, it went to the point where the TCP/IP accept socket function is unblocked by the client’s response.

Note that SSL-enabled applications usually have a set of directives related to the SSL environment, which are set by the user and are transferred to System SSL via the API. For instance, when setting up the z/OS http server to use SSL, a directive in the server configuration file specifies the location of the keys and certificates to be used by the server:

```
KeyFile racf_keyring_name SAF
```

When performing an SSL communication, the z/OS http server invokes System SSL functions and, in this example, is to pass this directive value as the value of environment variable GSK_KEYRING_FILE in the parameter list of the gsk_environment_open() System SSL function.

Upon invocation via the API, the System SSL code prepares the proper TCP messages, which are then sent by the application via regular TCP/IP sockets functions. In the same way the application receives TCP messages, which are exploited by the System SSL code, thus performing whatever exchanges are needed by the handshake sequence and the encryption/decryption of the transferred data. Optionally, the System SSL code can access an LDAP directory for CRL checking when receiving the partner’s certificate.

If the hardware cryptography is not enabled, or if the selected algorithms are not supported by the hardware crypto, System SSL performs the cryptographic algorithms with its own software encryption routines.

This figure is not showing the sessionID cache that is maintained in the application address space (unless the sysplex sessionID caching is in used, as explained below).

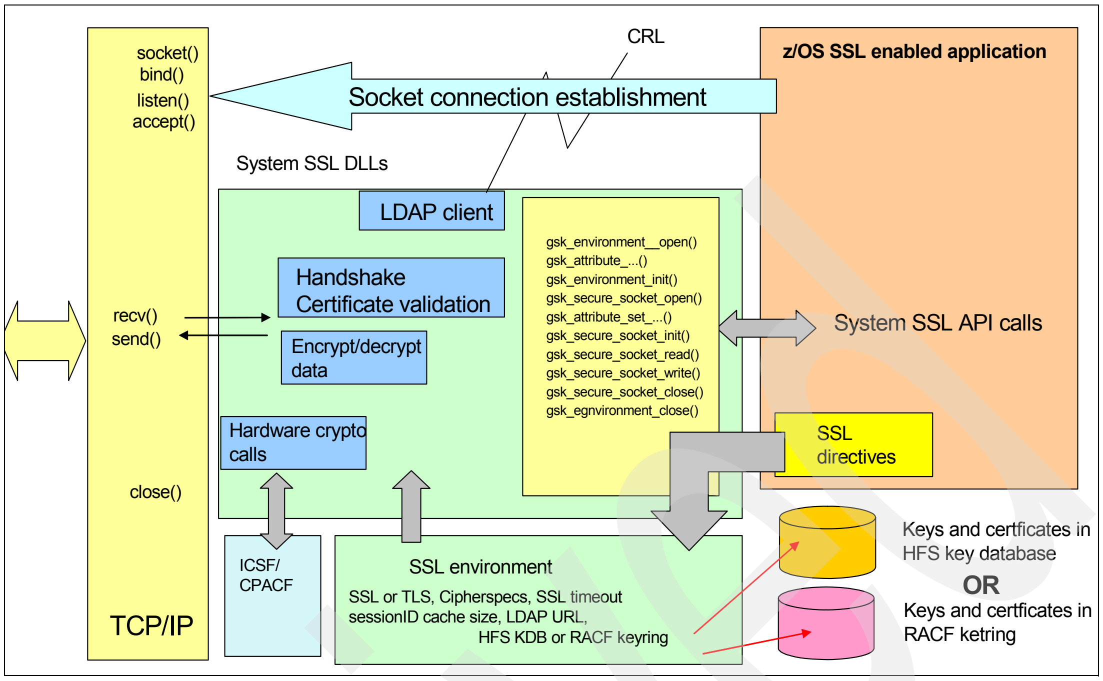


Figure 9-3 z/OS System SSL block diagram

A word on SessionID re-use - Sysplex SessionID caching

Session ID re-use, or *caching*, has been described, from the principle standpoint, in "The sessionID re-use" on page 210. SessionID caching is the main contributor in enhancing SSL/TLS performance by decreasing the frequency of full handshakes, and therefore the SSLtimeOut parameter must be thoroughly considered with respect to both the Security exposure of keeping the same secret key for an extended period of time and the potential performance increase. In z/OS System SSL the SSLtimeOut default value is 86400 seconds (24 hours).

The System SSL sysplex sessionID caching is available beginning with z/OS 1.4. The idea is that in configuration with workload dispatching, such as with a sysplex distributor, a re-connection and resumption of an SSL session can frequently occur on a server other than the original one, hence losing the benefit of being able to reuse the sessionID as it has been kept by the original server instance in its address space.

In a sysplex, the sessionIDs can be shared among servers via the GSKSRVR address space, which is an STC started in each sysplex member that is to share a sessionID with other members.

The local instance of GSKSRVR keeps the sessionID in a data space. When an SSL server in a sysplex member receives a non-locally known sessionID from a client, it enquires through the GSKSRV task if this sessionID is known from other GSKSRVRs in the sysplex. The inter GSKSRVR communications are performed via XCF. If the sessionID is known from another GSKSRVR, this GSKSRVR instance sends the corresponding pre-master secret, still using XCF, to the requesting instance. This is graphically shown in Figure 9-4 on page 217.

Note that the pre-master secret value is transmitted in clear as an intra-sysplex environment is considered as secure.

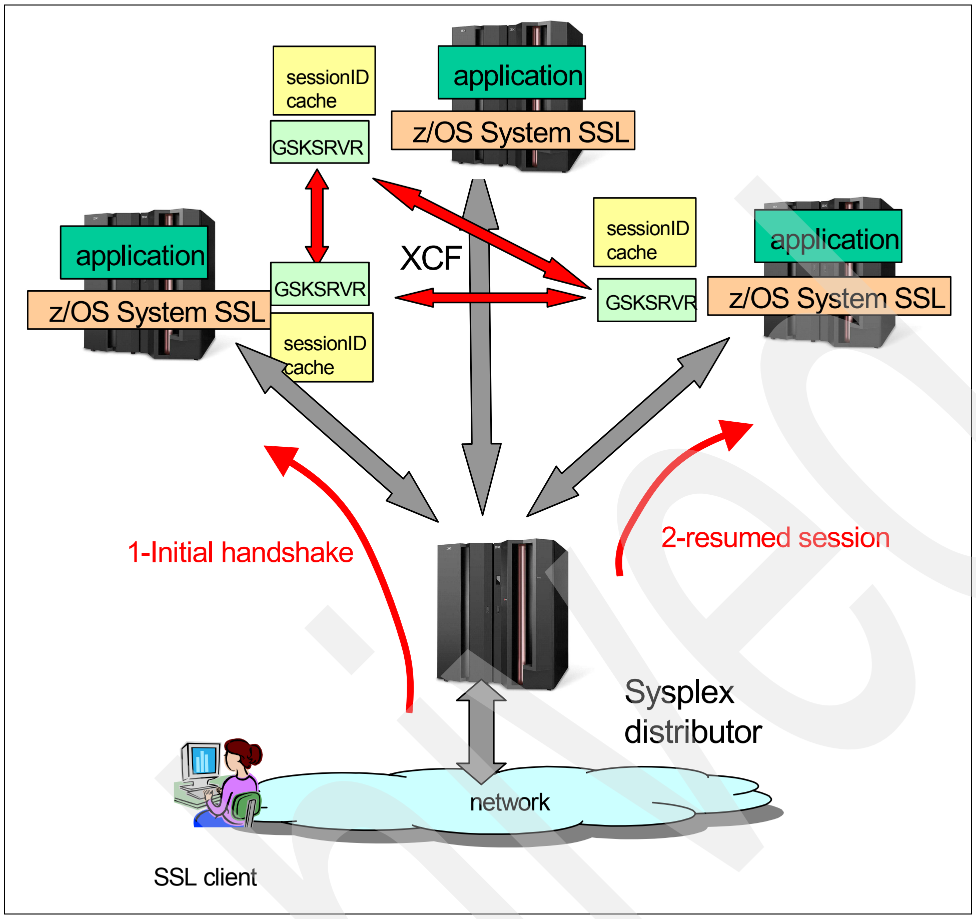


Figure 9-4 Sysplex session ID caching

9.4 z/OS System SSL and cryptography

In this section we discuss z/OS System SSL and cryptography.

9.4.1 Supported CipherSpecs

The following CipherSpecs are supported by z/OS System SSL at z/OS 1.6:

- SSL V2
  - 1 = 128-bit RC4 encryption with MD5 message authentication (128-bit secret key)
  - 2 = 128-bit RC4 export encryption with MD5 message authentication (40-bit secret key)
  - 3 = 128-bit RC2 encryption with MD5 message authentication (128-bit secret key)
  - 4 = 128-bit RC2 export encryption with MD5 message authentication (40-bit secret key)
  - 6 = 56-bit DES encryption with MD5 message authentication (56-bit secret key)
  - 7 = 168-bit Triple DES encryption with MD5 message authentication (168-bit secret key)

- The following SSL V3 cipher specifications are supported:
  - 00 = No encryption or message authentication and RSA key exchange
  - 01 = No encryption with MD5 message authentication and RSA key exchange
  - 02 = No encryption with SHA-1 message authentication and RSA key exchange
  - 03 = 40-bit RC4 encryption with MD5 message authentication and RSA key exchange
  - 04 = 128-bit RC4 encryption with MD5 message authentication and RSA key exchange
  - 05 = 128-bit RC4 encryption with SHA-1 message authentication and RSA key exchange
  - 06 = 40-bit RC2 encryption with MD5 message authentication and RSA key exchange
  - 09 = 56-bit DES encryption with SHA-1 message authentication and RSA key exchange
  - 0A = 168-bit Triple DES encryption with SHA-1 message authentication and RSA key exchange
  - 0C = 56-bit DES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with a DSS certificate
  - 0D = 168-bit Triple DES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with a DSS certificate
  - 0F = 56-bit DES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with an RSA certificate
  - 10 = 168-bit Triple DES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with an RSA certificate
  - 12 = 56-bit DES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with a DSS certificate
  - 13 = 168-bit Triple DES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with a DSS certificate
  - 15 = 56-bit DES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with an RSA certificate
  - 16 = 168-bit Triple DES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with an RSA certificate
  - 2F = 128-bit AES encryption with SHA-1 message authentication and RSA key exchange
  - 30 = 128-bit AES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with a DSS certificate
  - 31 = 128-bit AES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with an RSA certificate
  - 32 = 128-bit AES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with a DSS certificate
  - 33 = 128-bit AES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with an RSA certificate
  - 35 = 256-bit AES encryption with SHA-1 message authentication and RSA key exchange
  - 36 = 256-bit AES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with a DSS certificate

9.4.2 Invocation of the hardware cryptographic coprocessors

In this section we discuss the invocation of the hardware cryptographic coprocessors.

Utilization of the coprocessors

System SSL cipherSpecs, as of z/OS 1.6, allow for the use of symmetric data encryption using DES, Triple-DES, AES, RC2, and RC4, and the handshake asymmetric algorithms RSA, DSS, and DH (Diffie Hellman). Data integrity can be verified using the MD5 or SHA-1 algorithms.

ICSF support (that is, hardware support) is provided only for:

- DES or Triple-DES
- RSA
- SHA-1

System SSL senses if ICSF has started on the system and queries it to know which coprocessors types are available. On further request to proceed with cryptographic functions, System SSL will invoke hardware cryptography through ICSF if the algorithm to be used is supported by the coprocessors that are in operation.

ICSF can also be used to secure the RSA private key that System SSL uses on behalf of the SSL server or client application that calls it. This is available only when the keys are kept in the RACF database; in this case the private key is actually kept in the ICSF PKDS (Public Key Data Set) VSAM data set, where it is secured by being encrypted with the coprocessors asymmetric master key. This protection scheme of the RSA private key is further developed in “Cryptographic protection of the RSA private key” on page 221.

Using the CPACF on z990 and z890 systems

Part of System SSL has been rewritten to support the CP Assist for Cryptographic Functions (CPACF) when running on z990 and z890. CPACF is a z990/z890 standard feature that provides hardware assist for the DES or T-DES encryption of the data and for SHA-1 integrity checking as well. On 9672, z900, and z800, the standard cryptographic feature is the Cryptographic Coprocessor Facility (CCF). Note, however, that:

- CPACF does not provide all the functions provided by the CCF coprocessors.
- CPACF has to be enabled by installing FC 3863 on the system.

For complete details on the zSeries hardware cryptography implementation, refer to the redbook *IBM eServer zSeries 990 (z990) Cryptography Implementation*, SG24-7070.

When migrating to a z990 or z890 system, System SSL can be upgraded with the following PTFs to be able to use the CPACF:

- For OS/390 2.10: UA05142, UA06792
- For z/OS 1.2 and 1.3: UA05143, UA05299
- For z/OS 1.4 and 1.5: UA05144

When using a CPACF, the System SSL trace shows `Clear Key DES encryption …` as opposed to `Software DES encryption performed …` or `CCF DES encryption performed …`. More information about the System SSL trace is given in 9.8, “System SSL diagnostics facilities” on page 232.

DES or Triple-DES hardware support

With a system with the CPACF feature (that is, z990 and z890) System SSL proceeds in different ways depending on the z/OS release:

- For z/OS releases prior to z/OS 1.4, System SSL uses the ICSF services CSNBSYE (Symmetric Key Encipher) or CSNBSYD (Symmetric Key Decipher) to invoke the CPACF functions.
- At z/OS 1.4, System SSL still needs ICSF to query the cryptographic facilities installed (the Query facility service, CSFIQQF). It then directly calls the CPACF hardware on z990/z890.
- At z/OS 1.6, ICSF is no longer needed in order to determine what is available in the system. System SSL does its own query and calls the CPACF directly.

With other systems where the CCF is the standard cryptographic feature, System SSL calls the following ICSF services:

- CSFCKI - Clear key import
- CSFCKM - Multiple clear key import
- CSFDEC - Symmetric key decrypt
- CSFENC - Symmetric key encrypt

These functions are required since the CCF only deals with secure keys (that is, keys encrypted with the coprocessor master key) when performing symmetric algorithms. The key import services above are here just to encrypt the clear symmetric key that System SSL holds since the handshake completion, with the coprocessor master key.

SHA-1 support

On z900 or z800, System SSL always does SHA-1 in software.

On z990 or z890, System SSL does SHA-1 in software at z/OS 1.5 and earlier releases. Beginning with z/OS 1.6, System SSL calls the CPACF to perform the SHA-1 function in hardware.

RSA support

RSA cryptographic hardware assist requires one of the following coprocessors to be in operation on the system:

- CCF or PCICC for 9672, z900, and z800 systems (CCF always has to be enabled on these systems to support the PCICC.)
- PCICA for z900,z800,z990, or z890 systems (CCF always has to be enabled on z900 and z800 to support the PCICA.)
- PCIXCC for z990 and z890 systems
- Crypto Express2 for z990 and z890 systems

The ICSF services that are called are:

- CSFDSG - Digital signature generate
- CSFDSV - Digital signature verify
- CSFPKD - PKA decrypt
- CSFPKE - PKA encrypt
- CSFPKI - PKA key import

Important: In case of crypto hardware malfunction, System SSL can revert very silently to software encryption. No messages are being issued and only a System SSL trace would show that hardware encryption calls were just replaced by software encryption.

Cryptographic protection of the RSA private key

This optional function consists of keeping the RSA private key used by System SSL on behalf of the client or server application in an ICSF cryptographic data set: The Public Key Data Set (PKDS). Private keys kept in this data set are encrypted under the coprocessors master key, and therefore appear in clear only inside the secure enclosure of the coprocessor itself. The requirements to use this facility are:

- The RSA key pair has to be kept in the RACF database, as opposed to keeping it in an HFS file.
- One of the coprocessors listed above for the RSA encryption support, except for PCICA, must be in operation in the system. PCICA does not offer the services that this option requires, as it does not have a master key.

Figure 9-5 on page 222 is a graphical representation of the RSA private key protection. In this example an SSL server private key is protected in the PKDS and is used by the server during the handshake phase. A client application running in z/OS can also have the RSA private key protected. The key is used then for SSL client authentication.

In Figure 9-5 on page 222:

- The key pair is either generated in RACF with the « ICSF » » parameter, or the key pair is added in the RACF database, still with the « ICSF » parameter. The generation or importation process in that case stores the private key encrypted with the crypto master key in the PKDS, and stores in RACF the PKDS label of the key.
- When the private key is needed to perform the RSA algorithm, during the SSL handshake, System SSL finds out that the private key is not actually in the RACF database, but instead invokes ICSF to perform the RSA function and provide the PKDS key label found in the RACF database.

Very important: A private key once entered in the PKDS can never been extracted in clear again, by design of the cryptographic coprocessors. To be able to recover the private key implies having securely kept by other means a clear copy of the key.

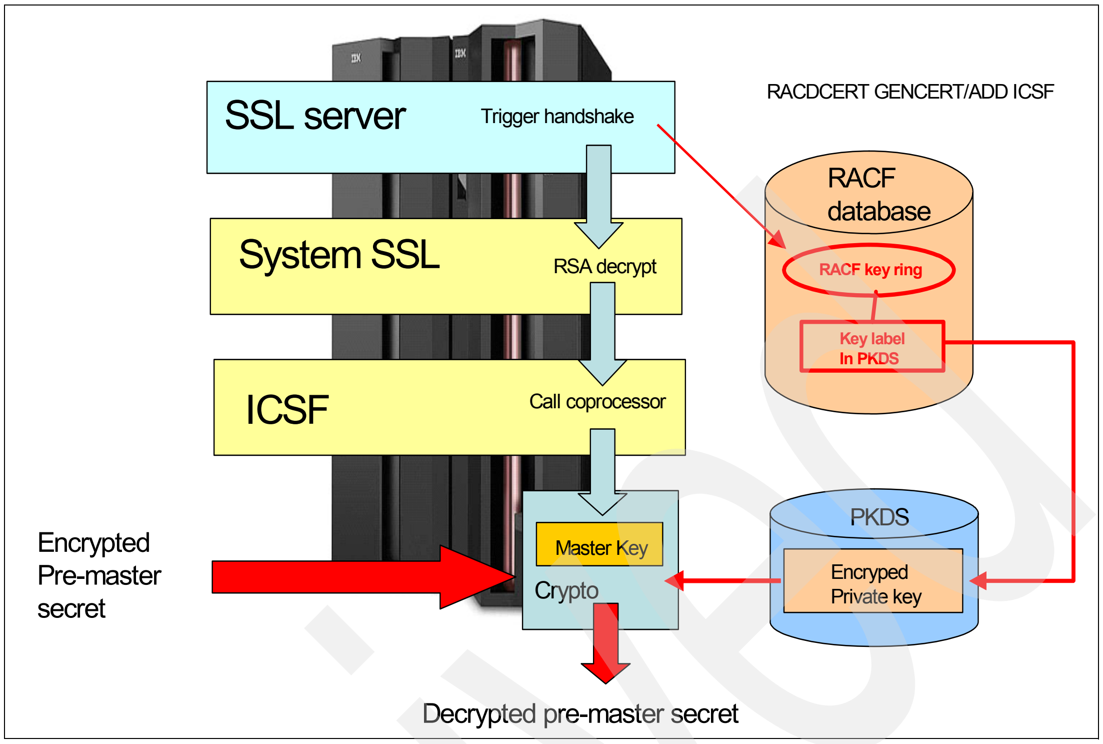


Figure 9-5 Protecting the RSA private key in the PKDS

9.5 Managing keys and certificates with gskkyman

As already mentioned, System SSL will get keys and certificates either from an HFS key database or from a RACF keyring; that is, from the RACF database. The choice is application or application deployment dependant. For example, WebSphere Application Server for z/OS only allows the use of RACF keyrings, whereas the z/OS IBM HTTP Server lets the user decide whether a RACF keyring or an HFS key database will be used.

This section is intended to be a reminder only of the use of the gskkyman keys and certificates management utility. Further details can be found in the z/OS bibliography in *z/OS Cryptographic Services System Secure Sockets Layer Programming*, SC24-5901.

Note that manuals for the specific SSL/TLS-enabled z/OS clients and servers explain, step-by-step, how to interact with gskkyman to build the required set of certificates and keys.

9.5.1 A reminder

gskkyman is a z/OS UNIX utility provided with System SSL to manage certificates and keys stored in HFS files called key databases. A key database is built by gskkyman and is populated at creation time with certificates of the most well-known certification authorities, so that the owning application can automatically validate any certificate it eventually receives that has been signed by one of these authorities.

Note: gskkyman cannot manage RACF keyrings.

Only RSA or DSA key pairs or certificates can be created or imported into a key database and given a label for retrieval when needed by System SSL on behalf of an SSL-enabled application. So, a key database can host certificates alone, that is, the stored record contains a certificate, and the imbedded public key, only, or the stored record contains both a private key and a public key, the latter still under the form of a digital certificate. These stored records are given a label at their creation. A certificate and its associated private key can be specified as the default certificate in the key database; that is, any certificate and key retrieval request that do not specify a label will get this default record returned.

Note: For the reader not familiar with the use of gskkyman, we review a certificate generation and signing process in "Signing a certificate in the gskkyman dialog" on page 225.

Access to the keys and certificates in the key database is protected by a password that the key database administrator has set and that is kept in a stash file for applications to be able to open the key database. The access to the HFS itself is based on the POSIX permission bits, and optionally on the z/OS HFS ACLs.

gskkyman is invoked in the z/OS UNIX shell and provides dialog options for:

- Basic X.509 certificate management
- Creation of HFS key database file
- Creation of self-signed certificates
- Creation of certificate requests
- Receiving of signed certificate requests
- Imports/exports of certificates, with and without associated private keys
- Storing the encrypted database password
- Setting default certificate

Note that gskkyman itself exploits the z/OS hardware cryptography if in operation on the system.

9.5.2 gskkyman enhancements at z/OS 1.4

Very briefly:

- gskkyman supports the latest level of the PKCS12 (Version 3, in addition to Version 1 previously) standard to import or export key pairs, and PKCS#7 format to export or import certificates. The latter is supported with a restriction, in that only one certificate can be imported or exported in PKCS#7 format (that is the first of the chain if dealing with a certificates chain).
- DSS key pairs and certificate requests can be created, with a 1024-bit DSA key.
- RSA key pairs can be created, with RSA key length of 1024 and 2048 bits.

9.5.3 gskkyman enhancements at z/OS 1.6

In z/OS 1.6 gskkyman has undergone further restructuring, resulting in the following new capabilities:

- Creation of certificates that contain subject alternate name extensions
- Creation of RSA certificates with public/private key sizes of 4096 bits
- Creation of fixed key pair parameters and Diffie-Hellman signed certificates using either RSA or DSS signing certificates
- Export/import of PKCS #7 file containing a chain of certificates
- Renewal of existing certificates

- Ability to sign certificates without leaving dialog mode instead of having to go in command line mode as it was the case before

Subject alternate name extension

Gskkyman in z/OS 1.6 allows us to add subject alternate name extensions to either self-signed certificates, signed certificates, or certificate requests. When a subject name has been specified in the creation dialog, the gskkyman program offers to specify subject alternate names as well. This is shown in Figure 9-6.

```
Enter 1 to specify subject alternate names or 0 to continue: 1

       Subject Alternate Name Type

   1 - Directory name (DN)
   2 - Domain name (DNS)
   3 - E-mail address (SMTP)
   4 - Network address (IP)
   5 - Uniform resource identifier (URI)

Select subject alternate name type (press ENTER if name is complete):
```

Figure 9-6 Specifying subject's alternate names with gskkyman

The options in the figure are reviewed below:

1. *Directory name* is an X.500 directory name; if this option is selected, an administrator can enter an X.500 distinguished name with the following attributes:
   - Common name (required)
   - Organizational unit (optional)
   - Organization (required)
   - City/Locality (optional)
   - State/Province (optional)
   - Country/Region (2 characters - required)
2. *Domain Name* consists of one or more qualifiers, separated by periods; for example:

   `cangate1.mkm.can.ibm.com`
3. *Email Address* is a familiar e-mail address with a user name and a domain name; for example:

   `yurykrit@ca.ibm.com`
4. *Network Address* may be in either the IPv4 format (nnn.nnn.nnn.nnn), or the IPv6 format (mmmm:mmmm:mmmm:mmmm:mmmm:mmmm:mmmm:mmmm).
5. *Uniform Resource Identifier* (URI) is the entity consisting of a schema name, a domain name, and a scheme-specific-portion; for example:

   `http://www.can.ibm.com/services/index.html`

The subject's alternate name information is also available to the System SSL APIs. For example, gsk_attribute_get_cert_info() has been enhanced to return the subject alternate name values.

Certificates with 4096-bit RSA keys

The key lengths now proposed by gskkyman for an RSA key pair generation are 1024, 2048, and 4096 bits. This applies to either CA (that is, self-signed) or user or server certificates.

Fixed key pair parameters and Diffie-Hellman signed certificates

This function allows us to create a server certificate to be used during an SSL handshake using a fixed Diffie-Hellman key exchange. Fixed Diffie-Hellman requires both sides of the exchange to be based off of the same generation parameters. In order for each side to use the same generation parameters, a key parameter file must be created to be used as input to the certificate being signed.

The key parameter file contains parameters for either a 1024-bit or 2048-bit Diffie-Hellman key.

Once the key parameter file has been created, the next step is to create the signed certificate using an existing certificate in the key database file to sign the server certificate. This is achieved via the gskkyman dialog by first selecting the signing certificate, then selecting a request to sign a user or server certificate, and specifying the previously created key parameter file as input to the certificate.

Import/export of certificates with certificate chains (PKCS#7)

The capability of importing or exporting in a certificate chain format was already available in gskkyman at z/OS 1.4; however, it was operating only on the first certificate in the chain. The full PKCS#7 chain support is now available in z/OS 1.6—that is, all certificates in the chain are being imported or exported.

Certificate renewals

This is the capability of creating a certificate request from an already existing certificate that is near expiration. The associated key pair and the information in the certificate that will be returned by the Certification Authority remain the same, except for the validity period, which is renewed by the CA and a new certificate serial number. The renewed certificate is then received in the key database to replace the current certificate. The process steps are:

- Option 1: “Manage keys and certificates” needs to be selected from the main menu. Then, a particular certificate is to be selected by the assigned number from the Key and Certificate list.
- This brings up the updated Key and Certificate Menu with the new Option 11: “Create a certificate renewal request.” After choosing this option, the gskkyman program prompts for a file name, which is to contain the renewal request. The resulting file is created in base64 encoded format and is to be sent to the signing CA.
- Upon the return of the renewed certificate, it is to be received through Option 5: “Receive Requested Certificate or a Renewal certificate” on the Key Management Menu.

Signing a certificate in the gskkyman dialog

We take this opportunity to review the steps to follow to create a signed user or server certificate.

1. We open our Key Database (KDB) HFS file in order to create our own Certification Authority (CA) certificate. The gskkyman Key Management Menu is shown in Figure 9-7 on page 226. We select option 6 in this menu, which leads us to the next step.

```
 Key Management Menu

       Database: /u/yury/yurykdb.kdb

   1 - Manage keys and certificates
   2 - Manage certificates
   3 - Manage certificate requests
   4 - Create new certificate request
   5 - Receive requested certificate or a renewal certificate
   6 - Create a self-signed certificate
   7 - Import a certificate
   8 - Import a certificate and a private key
   9 - Show the default key
  10 - Store database password
  11 - Show database record length

   0 - Exit program

Enter option number (press ENTER to return to previous menu):
 ===>
```

Figure 9-7 Main gskkyman Key Management Menu

2. We now create a CA certificate as shown in Figure 9-8. This results in having in the KDB an RSA key pair with keys of 1024-bit long and the public key being inserted in a self-signed certificate.

```
Enter option number (press ENTER to return to previous menu): 6

       Certificate Type

   1 - CA certificate with 1024-bit RSA key
   2 - CA certificate with 2048-bit RSA key
   3 - CA certificate with 4096-bit RSA key
   4 - CA certificate with 1024-bit DSA key
   5 - User or server certificate with 1024-bit RSA key
   6 - User or server certificate with 2048-bit RSA key
   7 - User or server certificate with 4096-bit RSA key
   8 - User or server certificate with 1024-bit DSA key

Select certificate type (press ENTER to return to menu): 1
Enter label (press ENTER to return to menu): Yury CA Cert
Enter subject name for certificate
  Common name (required): Yury CA Cert RSA 1024
  Organizational unit (optional): ITSO
  Organization (required): ibm
  City/Locality (optional): Montpellier
  State/Province (optional):
  Country/Region (2 characters - required): fr
Enter number of days certificate will be valid (default 365):

Enter 1 to specify subject alternate names or 0 to continue: 0

Please wait .....

Certificate created.
```

Figure 9-8 Creation of an installation CA certificate

3. We now return to the Main menu and choose Option 1: “Manage keys and certificates.” We select the CA certificate we just created. In Figure 9-9, our new CA certificate is selected and it will be used for the signing of a new user certificate by selecting the option 10: “Create a signed certificate and key.”

```
 Key and Certificate List

       Database: /u/yury/yurykdb.kdb

   1 - Yury CA Cert

   0 - Return to selection menu

Enter label number (ENTER to return to selection menu, p for previous list):
 ===> 1

       Key and Certificate Menu

       Label: Yury CA Cert

   1 - Show certificate information
   2 - Show key information
   3 - Set key as default
   4 - Set certificate trust status
   5 - Copy certificate and key to another database
   6 - Export certificate to a file
   7 - Export certificate and key to a file
   8 - Delete certificate and key
   9 - Change label
  10 - Create a signed certificate and key
  11 - Create a certificate renewal request

   0 - Exit program

Enter option number (press ENTER to return to previous menu):
 ===>
```

Figure 9-9 Updated Key and Certificate menu with the new Option 10

4. We select Option 10 and follow the dialog, which is shown in Figure 9-10 on page 228.

```
Enter option number (press ENTER to return to previous menu): 10

       Certificate Type

   1 - CA certificate with 1024-bit RSA key
   2 - CA certificate with 2048-bit RSA key
   3 - CA certificate with 4096-bit RSA key
   4 - CA certificate with 1024-bit DSA key
   5 - User or server certificate with 1024-bit RSA key
   6 - User or server certificate with 2048-bit RSA key
   7 - User or server certificate with 4096-bit RSA key
   8 - User or server certificate with 1024-bit DSA key
   9 - User or server certificate with 1024-bit Diffie-Hellman key
  10 - User or server certificate with 2048-bit Diffie-Hellman key

Select certificate type (press ENTER to return to menu): 5
Enter label (press ENTER to return to menu): Yury cert
Enter subject name for certificate
  Common name (required): Yury regular cert
  Organizational unit (optional): itso
  Organization (required): ibm
  City/Locality (optional): Montpellier
  State/Province (optional):
  Country/Region (2 characters - required): fr
Enter number of days certificate will be valid (default 365):

Enter 1 to specify subject alternate names or 0 to continue: 0

Please wait .....

Certificate created.

Press ENTER to continue.
```

Figure 9-10 Creation of a new user certificate, signed by the installation's CA certificate

If we list the newly created user certificate with the "Show certificate information" in the Key and certificate menu, the expected issuer and subject information is displayed by the utility, as shown in Figure 9-11 on page 229.

```
 Certificate Information

           Label: Yury cert
       Record ID: 17
Issuer Record ID: 16
         Trusted: Yes
         Version: 3
   Serial number: 4135a6540000ed0b
     Issuer name: Yury CA Cert RSA 1024
                  ITSO
                  ibm
                  Montpellier
                  FR
    Subject name: Yury regular cert
                  itso
                  ibm
                  Montpellier
                  FR
  Effective date: 2004/09/01
 Expiration date: 2005/09/01
```

Figure 9-11 New user certificate, signed by the installation's CA certificate

9.6 Managing keys and certificates with RACDCERT command

Here we just provide a reminder of the use of the RACDCERT command in the context of setting up keyrings for applications calling System SSL. Further details on the new certificate management functions in RACF at z/OS 1.6 can be found in 2.5, "RACF digital certificates handling enhancements" on page 42.

The reference manuals to be used in the z/OS bibliography are *z/OS Security Server RACF Command Language Reference*, SA22-7687, and *z/OS Security Server RACF Security Administrator's Guide*, SA22-7683.

Alternatively to using HFS files to store keys and certificates, and actually the IBM recommended approach, keys and certificates can be managed in the RACF (or equivalent) database using the RACDCERT RACF administrative command. Usually the SSL/TLS-enabled application has a configuration directive that specifies either a key database HFS file or a SAF keyring as the repository for keys end certificates. This information is propagated in the parameter list when the application calls System SSL.

Keys and certificates are aggregated into keyrings. Certificates are actually kept in RACF profiles in the DIGTCERT class, and keyrings are profiles in the DIGTRING class. Certificates and keyrings are owned by RACF user IDs.

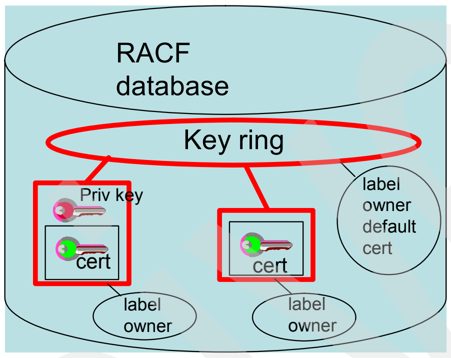
In order to get access to certificates and keyrings, applications must be permitted to RACF profiles in the FACILITY class with names IRR.DIGTCERT.<function>. Where <function> can be, for example, LIST to get access to a certificate, and LISTRING to get access to a keyring. An example of a creation of certificates and keyring is shown in Figure 9-12 on page 230.

RACF keyrings can be shared between SSL-enabled applications, specifying who is the owner of the keyring and the name of the keyring; typically this is indicated to the application by a parameter like *owner_name/keyring_name.* The using user ID, usually the application user ID, must be permitted with at least UPDATE access to the IRR.DIGTCERT.LISTRING RACF profile in the FACILITY class.

Notes:

1. Although RACF user IDs are owners of certificates or keyrings, they still need to be permitted to the IRR.DIGTCERT.<function> profiles.
2. Although DIGTCERT and DIGTRING profiles are general resources profiles they must be managed via the RACDCERT command. The RDEFINE and RALTER commands yield unexpected results with these profiles
3. Applications are getting to certificates and keyrings via the IRRSDL00 RACF callable service.

Figure 9-12 is an example of the creation of certificates and keyring in the RACF database.



Creation of a Certification Authority RSA key pair and self-signed certificate
```
RACDCERT CERTAUTH GENCERT SUBJECTSDN(CN('RACF CA')O('PSSCRACF')C('FR'))WITHLABEL('RACF CA')
```

Creation of a server RSA key pair and a signed certificate
```
RACDCERT ID(WEBSRV) GENCERT SUBJECTSDN(CN('10.11.12.13') O('MOPPSSC') C('FR'))
   WITHLABEL('webserver') SIGNWITH(CERTAUTH LABEL('RACF CA'))
```

Creation of a server keyring
```
RACDCERT ID(WEBSRV) ADDRING(webring)
```

Connection of required certificates to the keyring
```
RACDCERT ID(WEBSRV) CONNECT(CERTAUTH LABEL('RACF CA') RING(webring))
RACDCERT ID(WEBSRV) CONNECT(LABEL('webserver') RING(webring)DEFAULT)
```

Permission for the server to use its own certificate and keyring
```
PE IRR.DIGTCERT.LIST CL(FACILITY) ID(WEBSRV) ACC(READ)
PE IRR.DIGTCERT.LISTRING CL(FACILITY) ID(WEBSRV) ACC(READ)
```

Figure 9-12 Reminder: RACF certificates and keyrings

As of the writing of this document, RACF supports key lengths of 2048 bits maximum.

Although keyrings and certificates can be shared between applications, an application has access to a private key only if it is the owner of the DIGTCERT profile that contains this key. With the exception of private keys in the RACF database, which are declared to belong to a Certification Authority (CERTAUTH) or to a Site (SITE), these private keys can be shared between users with proper RACF privileges (see the *z/OS Security Server RACF Security Administrator's Guide*, SA22-7683).

9.7 System SSL Certificate Management Services

These are services that were previously internally used by System SSL and gskkyman, and that have been externalized for use by client applications, beginning with z/OS 1.4. There are three families of APIs.

9.7.1 Certificate database manipulation API

This API gives application writers the ability to create and manage their key databases through their own programs, as opposed to having an administrator using the gskkyman dialog. This provides the required functions to automate certificate management. Note that RACF keyrings can also be addressed by the functions part of this API; however, accesses to the RACF keyrings are constrained to read-only kinds of access. Examples of functions in this API are:

- gsk_open_database(), gsk_open_keyring()
- gsk_get_default_key()
- gsk_get_record_by_label()
- ...

9.7.2 Certificate database exploitation API

This API gives application writers the interfaces needed to create, maintain, and use certificates and keys stored in key database and RACF keyrings. Note that the functions permitted for the RACF keyring are read-only kind of functions. These functions do not provide for alteration of the RACF database contents as one could do with the RACDCERT command.

Examples of functions in this API are:

- gsk_create_self_signed_certificate()
- gsk_create_certificate_request()
- gsk_create_signed_crl()
- gsk_validate_certificate()
- ....

9.7.3 PKCS #7 message API

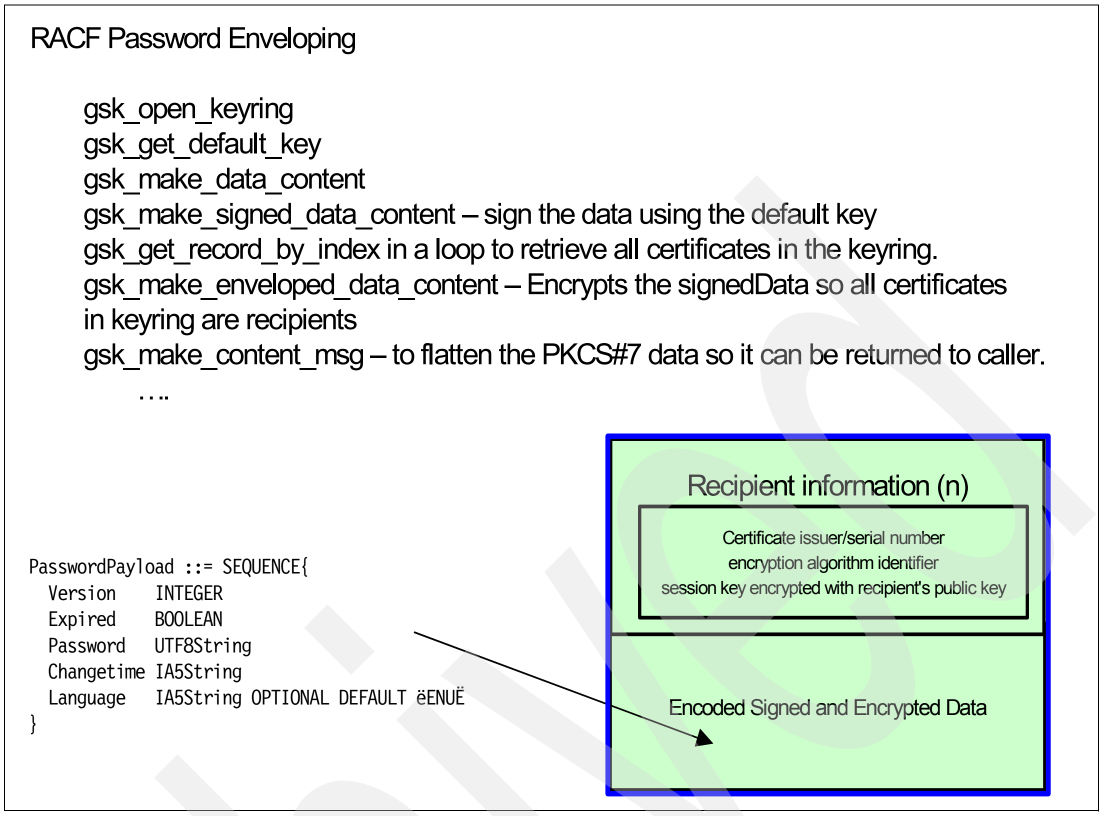
PKCS#7 is a de facto standard for formatting encrypted and signed messages. It is also used to package certificates or chains of certificates for exchanges between certificates repositories.

This API gives application writers the interfaces needed to build and process messages in the PKCS#7 standard (Cryptographic Message Syntax). Actually, this API is used by the RACF Password Enveloping function.

Examples of functions in this API are:

- gsk_make_signed_ data_content()
- gsk_make_enveloped_data_content()
- gsk_read_signed_data_content()
- gsk_read_enveloped_data_content()

Figure 9-13 on page 232 describes the PKCS#7 functions offered by System SSL as used by the RACF Password Enveloping function.


RACF Password Enveloping

gsk_open_keyring
gsk_get_default_key
gsk_make_data_content
gsk_make_signed_data_content – sign the data using the default key
gsk_get_record_by_index in a loop to retrieve all certificates in the keyring.
gsk_make_enveloped_data_content – Encrypts the signedData so all certificates in keyring are recipients
gsk_make_content_msg – to flatten the PKCS#7 data so it can be returned to caller.
….

```
PasswordPayload ::= SEQUENCE{
  Version    INTEGER
  Expired    BOOLEAN
  Password   UTF8String
  Changetime IA5String
  Language   IA5String OPTIONAL DEFAULT ëENUË
}
```

Recipient information (n)
Certificate issuer/serial number
encryption algorithm identifier
session key encrypted with recipient's public key
Encoded Signed and Encrypted Data

Figure 9-13 RACF Password Enveloping

9.8 System SSL diagnostics facilities

In this section we review System SSL diagnostics facilities.

9.8.1 High-level trace

A quite useful high-level trace can be started by setting up environment variables in the application that calls System SSL. These are:

- GSK_TRACE_FILE=raw_trace_filename, which indicates what HFS file is to receive the trace records. The default file is /tmp/gskssl.%.trc (the current process identifier is included as part of the trace file name when the name contains a percent sign).
- GSK_TRACE=trace_level, which specifies the desired level of tracing. The following tracing levels are available:
  - 0x01 = Trace function entry
  - 0x02 = Trace function exit
  - 0x04 = Trace errors
  - 0x08 = Include informational messages
  - 0x10 = Include EBCDIC data dumps
  - 0x20 = Include ASCII data dumps

The raw trace received in the file specified by GSK_TRACE_FILE has to be processed to become a readable trace. The following script command has to be invoked in the z/OS UNIX shell:

```
gsktrace raw_trace_filename > formatted_trace_filename
```

An example of the readable trace is given in Figure 9-14. These are three excerpts of an SSL trace where we can see that:

- Although ICSF is active on the system, no Master Key has been installed, yet in the crypto coprocessors render them non-operational for System SSL.
- A certificate is read out of a RACF keyring called WASKeyring.
- A DES software encryption is performed.

As already mentioned, there is no System SSL message issued when hardware encryption is not in operation. This trace actually provides the level of detail needed to understand whether the hardware cryptography is being used.

```
03/09/2005-18:30:15 Thd-0 INFO gsk_svc_init(): System SSL Version 3, Release 14, Service level
OA06156
03/09/2005-18:30:15 Thd-0 INFO gsk_svc_init(): LE runtime level 0x41050000
03/09/2005-18:30:15 Thd-0 INFO gsk_svc_init(): STDOUT handle=-1, STDERR handle=-1, TRACE handle=264
03/09/2005-18:30:15 Thd-0 INFO gsk_dll_init_once(): Using variant character table for code set
IBM-1047
03/09/2005-18:30:15 Thd-0 INFO gsk_dll_init_once(): Using local code page IBM-1047
03/09/2005-18:30:15 Thd-0 INFO gsk_dll_init_once(): Using ISO8859-1 for TELETEX string
03/09/2005-18:30:15 Thd-0 INFO gsk_dll_init_once(): 64-bit encryption enabled
03/09/2005-18:30:15 Thd-0 INFO gsk_dll_init_once(): 128-bit encryption enabled
03/09/2005-18:30:15 Thd-0 INFO gsk_dll_init_once(): Triple DES encryption enabled
03/09/2005-18:30:15 Thd-0 INFO crypto_init(): ICSF FMID is HCR7708
03/09/2005-18:30:15 Thd-0 INFO crypto_init(): No CCF with a valid master key is available
03/09/2005-18:30:15 Thd-0 INFO crypto_init(): DES hardware support is not available
03/09/2005-18:30:15 Thd-0 INFO crypto_init(): DES3 hardware support is not available
03/09/2005-18:30:15 Thd-0 INFO crypto_init(): PCI cryptographic coprocessor is not available
03/09/2005-18:30:15 Thd-0 INFO crypto_init(): PCI cryptographic accelerator is not available
03/09/2005-18:30:15 Thd-0 INFO crypto_init(): Public key hardware support is not available
....
03/09/2005-18:30:15 Thd-0 ENTRY gsk_open_keyring(): ---> Keyring 'WASKeyring'
03/09/2005-18:30:15 Thd-0 EBCDIC gsk_open_keyring(): IRRSDL00 output
        00000000:  1d9e56e0 00000008 00000001 0000027d  *...\..........'*
        00000010:  2f5aa8d8 0000027b 1ef47af0 00000001  *.!yQ...#.4:0....*
        00000020:  00000400 00000015 1d9e56f4 07e6c4f8  *...........4.WD8*
        00000030:  c3f1f1e4 40000000 00000039 1ef47f00  *C11U ........4".*
        00000040:  0000003b 1dbab3b8                    *........        *
03/09/2005-18:30:15 Thd-0 INFO gsk_open_keyring(): Record 'DefaultWASCert.CLD811' is the default
key
03/09/2005-18:30:15 Thd-0 INFO gsk_open_keyring(): Identifier 1 assigned to 'DefaultWASCert.CLD811'
03/09/2005-18:30:15 Thd-0 ENTRY gsk_decode_certificate(): --->
03/09/2005-18:30:15 Thd-0 ASCII gsk_decode_certificate(): Encoded X.509 certificate
        00000000:  30820279 308201e2 a0030201 02020103  *0..y0...........*
        00000010:  300d0609 2a864886 f70d0101 05050030  *0...*.H........0*
        00000020:  47311a30 18060355 040b1311 57656253  *G1.0...U....WebS*
        00000030:  70686572 6520666f 72207a4f 53312930  *phere for zOS1)0*
        00000040:  27060355 04031320 57415320 43657274  *'..U... WAS Cert*
        00000050:  41757468 20666f72 20536563 75726974  *Auth for Securit*
        00000060:  7920446f 6d61696e 301e170d 30343039  *y Domain0...0409*
        00000070:  30383232 30303030 5a170d31 30313233  *08220000Z..10123*
....
03/09/2005-18:30:15 Thd-0 INFO crypto_des_encrypt(): Software DES encryption performed for 640
bytes
```

Figure 9-14 System SSL high-level trace

9.8.2 System SSL Component Trace

A System SSL Component Trace can be collected via the GSKSRVR started task. This started task, also called the SSL Started Task, provides sysplex session ID caching as well.

Component trace records can be written to a trace external writer or kept in an in-storage trace buffer that is part of the GSKSRVR address space. The Component Trace can be started before the application to be traced is started or while the application is running, through the usual TRACE command at the system console:

```
TRACE CT,ON,COMP=GSKSRVR
R n,JOBNAME=(CS390IP,DB1G),OPTIONS=(LEVEL=15),WTR=GSKWTR,END
```

Where LEVEL specifies the trace options:

- 0x01 = Trace function entry
- 0x02 = Trace function exit
- 0x04 = Trace errors
- 0x08 = Include informational messages
- 0x10 = Include EBCDIC data dumps
- 0x20 = Include ASCII data dumps

IPCS CTRACE is used to format and display the trace records.

10

z/OS OpenSSH

In this chapter we describe OpenSSH, which is a function delivered in the IBM Ported Tools for z/OS unpriced program product (Program Number 5655-M23). OpenSSH is the open version of the very popular UNIX utility Secure Shell.

Note: OpenSSH is a new program product for z/OS 1.4. Note, however, that a ported version of OpenSSH is available as-is on the z/OS Unix Tools & Toys page at:

`http://www-1.ibm.com/servers/eserver/zseries/zos/unix/bpxa1toy.html`

This previous version should be removed prior to installing the IBM Ported Tools version.

10.1 SSH and OpenSSH

Secure Shell (SSH) is intended to be a secure version of the traditional BSD UNIX utilities: rlogin (remote login), rsh (remote shell), and rcp (remote copy).

- rlogin starts a terminal session from the local shell a remote host specified as host. The remote host must be running a rlogind service for rlogin to connect to.
- rsh executes a command, or provides remote shell access, on the specified remote host, which must be running the rshd service.
- rcp copies files between machines. Each file or directory argument is either a remote file name or a local file name

SSH is actually a client-server protocol and a suite of connectivity tools. The ssh client program is a secure remote login program—that is, a secure alternative to rlogin or rsh. The SSH suite also provides sftp, which is a secure version of ftp, and rcp rcp, a secure version of rcp. The client program requests a connection to the sshd secure remote login daemon running in the target host. The sshd daemon handles key exchange, encryption, authentication of both server and user, command execution, and data exchange. By default sshd listens to port 22 for incoming connection requests.

A very schematic view of SSH is given in Figure 10-1 on page 237.

The ssh command

Although the ssh command provides many command-line options, you can use it as though you were using the rsh command. If you specify a command argument, the ssh command remotely executes the specified command on the remote server and returns the command output on the local controlling terminal. If you do not specify the command to be remotely executed as a command line then you log in to the remote system through the secure connection established by OpenSSH.

scp

The secure copy gives you the ability to copy files between two hosts using a secure connection. Note that scp can be invoked to run between two hosts remote from the ssh client.

sftp

If you are familiar with the user interface of the ftp command, you can use the sftp command to securely transfer files. When you invoke the sftp command you are prompted to enter subcommands very much alike the ftp subcommands.

Attention: ftp and sftp are different and non-interoperable protocols.

Another interesting capability of SSH is to be able to use the encrypted tunnel to carry another TCP protocol such as http, telnet, TN3270, etc. This capability is referred to as TCP Port Forwarding and is discussed later in this chapter.

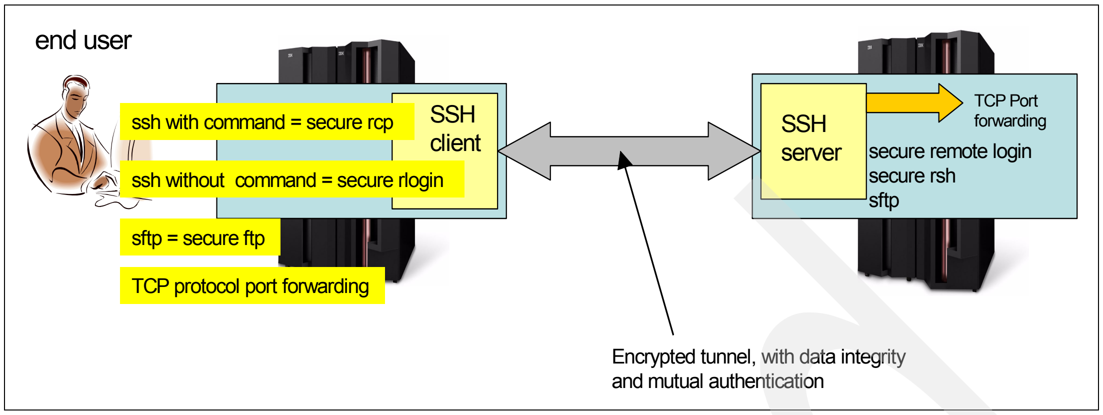

end user
ssh with command = secure rcp
ssh without command = secure rlogin
sftp = secure ftp
TCP protocol port forwarding
SSH client
SSH server
TCP Port forwarding
secure remote login
secure rsh
sftp
Encrypted tunnel, with data integrity and mutual authentication

Figure 10-1 SSH overview

SSH-provided protection

SSH is intended to provide protection against:

- Eavesdropping
- Name service and IP spoofing
- Connection hijacking
- Man-in-the-middle attacks
- Insertion attacks

However, SSH does *not* provide protection against:

- Password cracking
- IP and TCP packet or protocol level attacks
- Traffic analysis
- Covert channels (A covert channel is the unexpected capability in a system, from the design standpoint, to transfer information from one user to another.)

SSH protocol versions

The SSH protocol is available in two versions: Version 1 and version 2. Version 2 (SSH-2) is a proposed set of standards from the Internet Engineering Task Force called the *SECH protocol*, that can be found at:

`http://www.ietf.org/html.charters/secsh-charter.html`

The SSH1 protocol has been deprecated, as it is no longer felt to be secure enough. Comparing Version 1 and Version 2 yields the following differences:

- SSH2 is made of separate, layered protocol, whereas SSH1 is monolithic.
- SSH2 uses strong crypto integrity checking; SSH1 uses the weaker CRC-32 check.
- SSH2 supports many numbers of session channels per connection; SSH1 supports one session channel per connection.
- SSH2 provides for negotiation of crypto algorithms; SSH1 only allows negotiating on the bulk cipher; all other algorithms are fixed.
- SSH2 supports the RSA and DSA public keys; SSH1 supports only RSA public keys.

Note: The SSH1 and SSH2 protocols are not compatible. From now on, unless specifically stated, we will implicitly refer to SSH2 only.

10.2 OpenSSH

OpenSSH is primarily developed by the OpenBSD Project (`http://www.openssh.com/`), and its first inclusion into an operating system was in OpenBSD 2.6. The software is developed outside the USA, using code from roughly 10 countries.

OpenSSH is a derivative of the original and free ssh 1.2.12 release by Tatu Ylonen. Aaron Campbell, Bob Beck, Markus Friedl, Niels Provos, Theo de Raadt, and Dug Song removed many bugs, re-added newer features, and created OpenSSH. Markus Friedl contributed the support for SSH protocol Versions 1.5 and 2.0.

10.2.1 OpenSSH principles of operation

OpenSSH, as does SSH, runs on top of the TCP/IP stack, provides security at the application layer, and can be thought of as a protocol with three layered components, as illustrated in Figure 10-2:

- The transport layer, which provides algorithm negotiation and key exchange. The key exchange includes server authentication and results in a cryptographically secured connection: It provides integrity, confidentiality, and optional compression of data.
- The user authentication layer uses the established connection and relies on the services provided by the transport layer. It provides several mechanisms for user authentication. These include traditional password authentication as well as public-key or host-based authentication mechanisms.
- The connection layer multiplexes many different concurrent channels over the authenticated connection and allows tunneling of login sessions and TCP-forwarding. It provides a flow control service for these channels. Additionally, various channel-specific options can be negotiated.

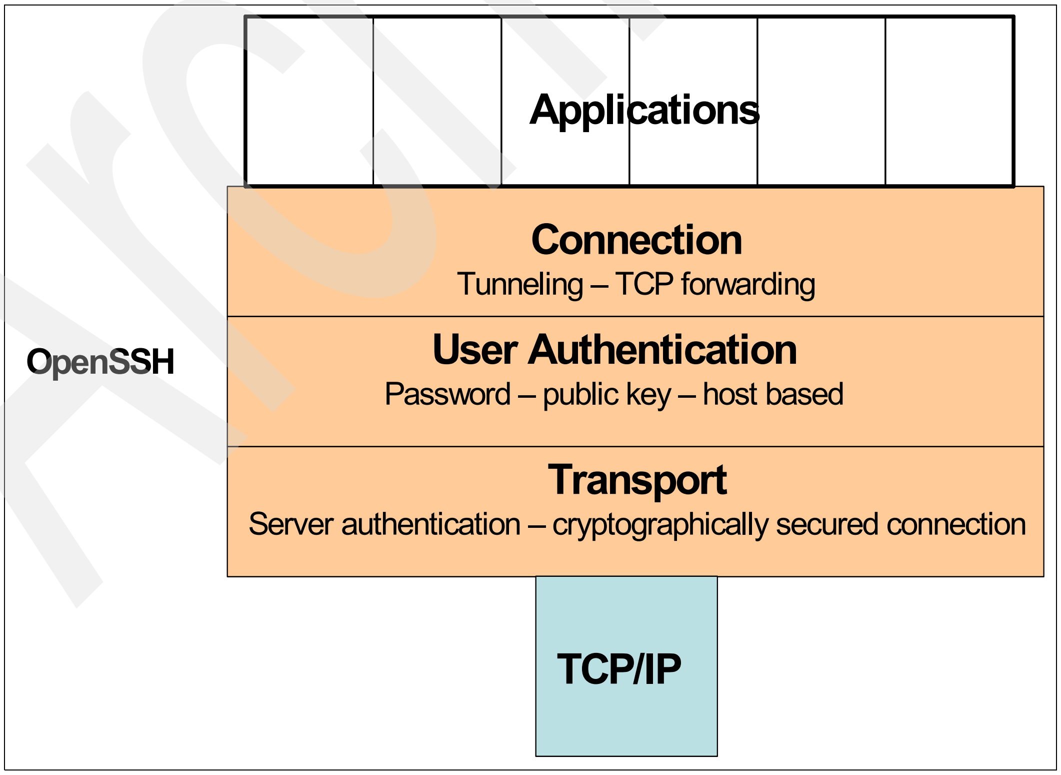

Applications
Connection
Tunneling – TCP forwarding
OpenSSH
User Authentication
Password – public key – host based
Transport
Server authentication – cryptographically secured connection
TCP/IP

Figure 10-2 The OpenSSH layered structure

Figure 10-3 on page 240 summarizes the interactions between an OpenSSH client and daemon. Note that a secure channel is established after authentication of the server (the host), then the communication proceeds with client-to-server authentication.

- Each host has a host-specific key (RSA or DSA) used to identify the host. Whenever a client connects, the server responds with its public host key. The client compares the RSA host key against its own database to verify if this key value is recorded as being owned by this very host. Packets sent by the host will be signed using the corresponding private key. Then a Diffie-Hellman key agreement exchange occurs that results in establishing a shared session key and a session number.
- The rest of the session is encrypted using a symmetric cipher. The ciphers currently supported are 128-bit AES, Blowfish, 3DES, CAST128, Arcfour, 192-bit AES, or 256-bit AES. The client selects the encryption algorithm to use from those offered by the server. Additionally, session integrity is provided through a cryptographic message authentication code (hmac-sha1 or hmac-md5).
- The client then authenticates using one of these negotiated authentication methods:
  - Public Key authentication (RSA or DSA)
  - Hosts file at the server

    This method is based on the traditional UNIX .rhosts and .shosts files. It is an inherently insecure authentication mechanism.
  - password
  - challenge-response (not supported on z/OS)
- Then the commands are executed and the data are exchanged.

The following types of keys are therefore involved in authentication and protection of the secure channel:

- Host key - An asymmetric key used by the server to provide the server identity.
- Session key - A dynamically generated symmetric key for encrypting the communication.
- User key - An asymmetric key used by the client to prove a user identity.

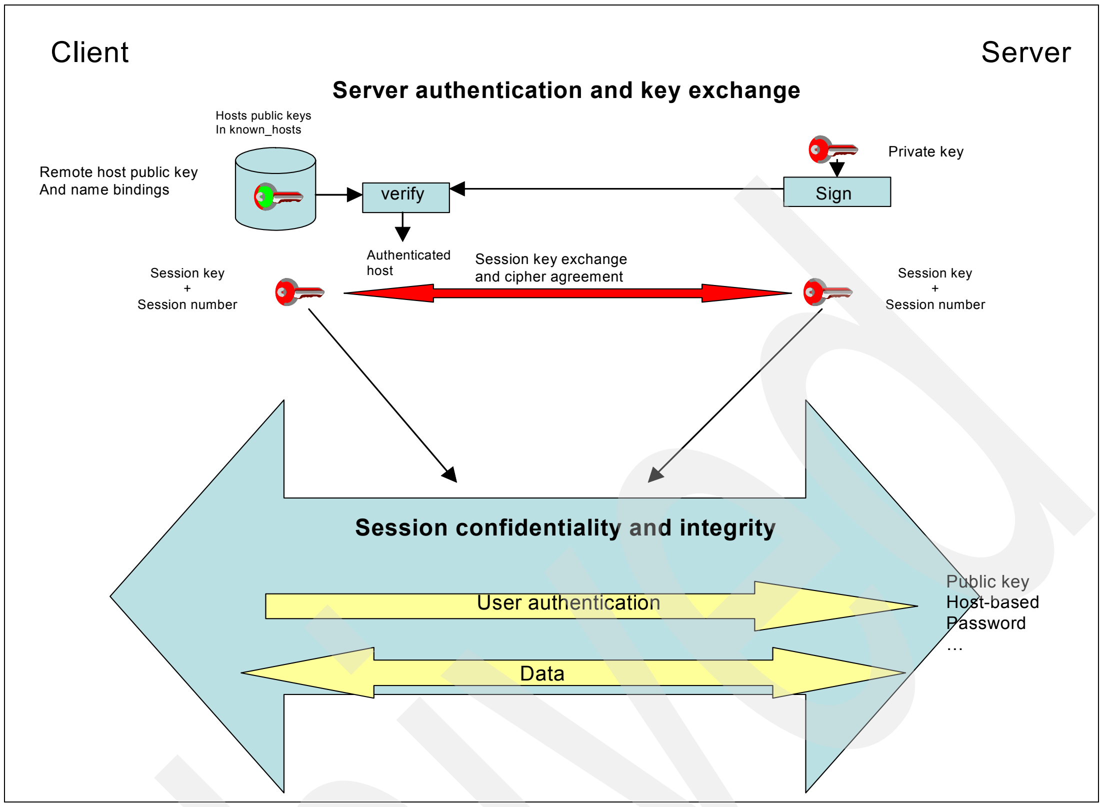


Figure 10-3 OpenSSH protocol sequences

10.2.2 Other OpenSSH features

Other OpenSSH features are discussed in this section.

Authentication agent

When the user uses asymmetric keys for authentication, the private key is usually kept protected at the client systems (that is, encrypted) using a passphrase that must be entered whenever the private key must be accessed during the authentication process. The SSH Authentication Agent, ssh-agent, is a program coming with z/OS OpenSSH that, when used, holds the authentication private keys and avoids having the end user entering the passphrase. The ssh-agent is started at the beginning of a session and is used automatically for RSA or DSA authentication when logging onto other machines using ssh. The agent stores the authentication private keys in memory.

After invoking the authentication agent, you have to load your private key into the agent using the ssh-add command.

Note: Although it is convenient to use, you should be aware of the potential security risk while you are using the authentication agent. If a malicious user takes over your terminal, he can remotely log in to any system without a prompted passphrase. You should terminate the authentication agent before you leave your terminal.

Privilege separation

Privilege separation has been implemented in OpenSSH as a means of restricting the effects of coding error or code compromise. OpenSSH has a monitor process running with superuser (UID(0)) authority, whereas the process in charge of authenticating a new request is running under the so-called *privilege separation user* identity with very low privileges in the system. When the request has been duly authenticated, it then runs in a process with the authenticated user's privileges.

This is described in Figure 10-4. The sequence of events is:

1. The privileged OpenSSH process, with superuser privileges, is listening on port 22 by default and is waiting for a new request.
2. When receiving a new request, the OpenSSH privileged process forks an unprivileged child, running with the privileges of the privilege separation user, to handle the new connection request. The unprivileged process requests the privileged process to proceed with user authentication. Any code compromise or error in this phase would lead to having the low privilege user only attempt unexpected actions.
3. When user authentication has been successfully performed, a new child process is forked, which runs with the authenticated user's privileges.

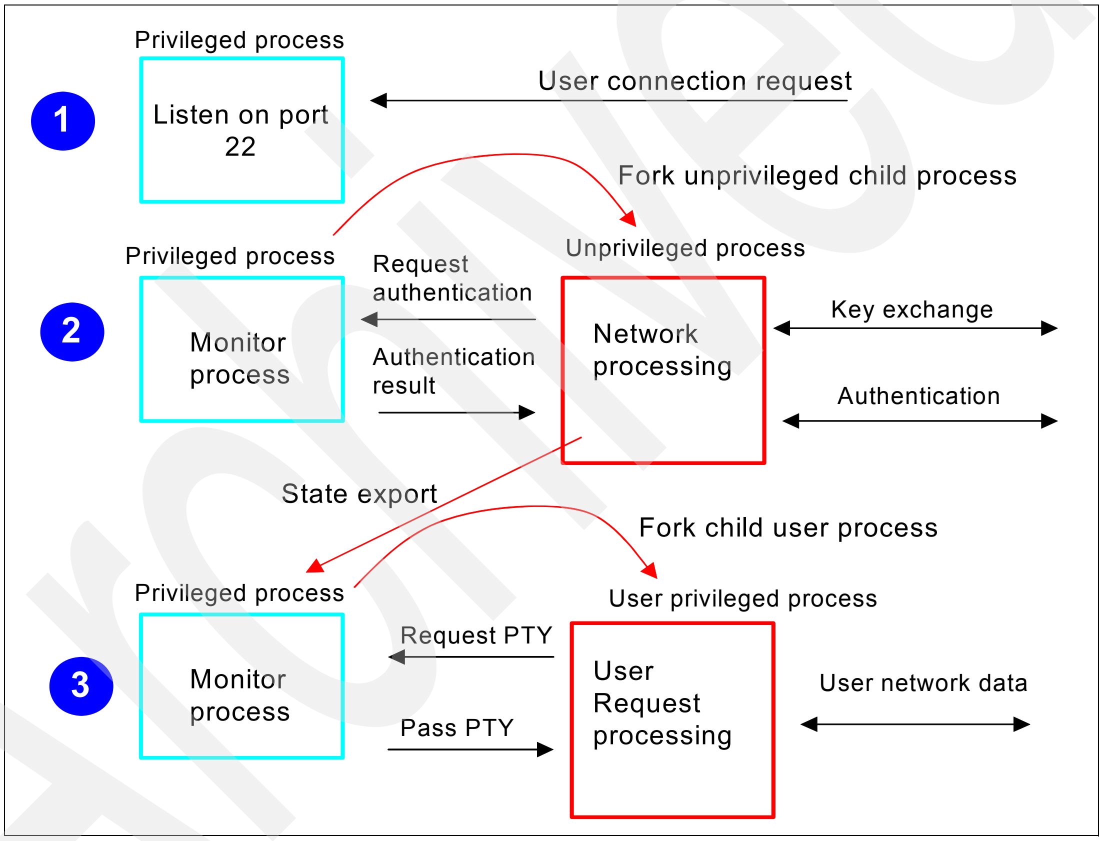


Figure 10-4 Privilege separation

SSH TCP/IP and X11 forwarding

Here we discuss SSH TCP/IP and X11 forwarding.

TCP/IP forwarding

TCP/IP forwarding is a powerful function that allows you to securely establish TCP connections between local and remote hosts by forwarding connection ports. Although initially intended for typical UNIX TCP-based protocols, such as SMTP and POP3, other protocols like http and ftp can exploit TCP/IP forwarding using the secure connection.

TCP/IP forwarding is classified into two modes according to where the port is forwarded from:

- Local forwarding - A connection is forwarded from the client host to the remote server host.
- Remote forwarding - A connection is forwarded from the remote server host to the local client host.

Figure 10-5 illustrates the concept of local TCP/IP forwarding. You want to connect an application client process (shown as A) on the client to an application server process (shown as B) on the application server. The application client uses the port specified as local port and the application server is listening on the port specified as remote port.

Before invoking the application client process, you have to establish a secure connection (shown as C) using the ssh command between the client and the SSH server. The ssh client process is instructed to enable a local port forwarding upon invocation, allowing you to connect the client application process to the ssh client (D), which is to forward the communication through the tunnel. An example of the command syntax to establish local forwarding at the OpenSSH client is:

```
ssh -L port_a:srv_x:port_y
```

This results in the interactions shown in Figure 10-5. The OpenSSH client then binds to port_a at the local IP address. The client application is told to connect to port_a at the local IP address, therefore connecting to the OpenSSH client using the forwarded port. This translates into a tunnel established to carry and secure the communication down to the OpenSSH server. The OpenSSH server in turn connects to the target host at srv_x, port_y and completes the communication forwarding.

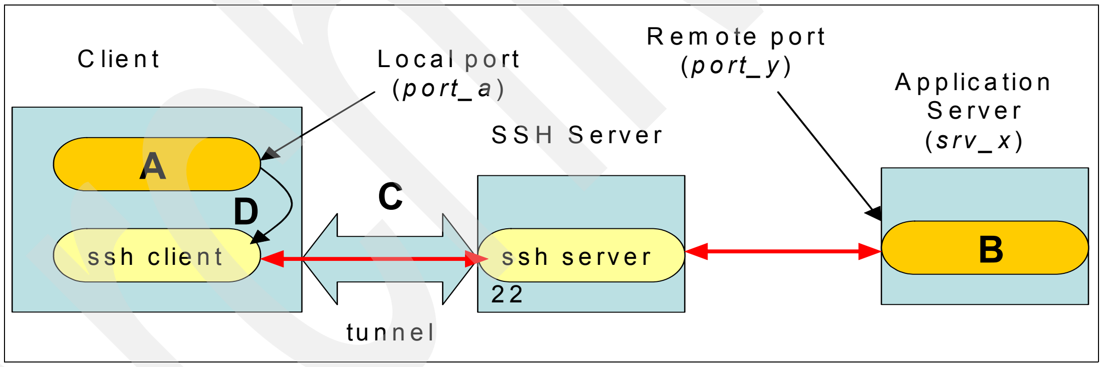


Figure 10-5 SSH TCP/IP local forwarding

Important: Note that port forwarding can be performed only after a remote login session has been opened from the SSH client to the remote SSH server.

X11 forwarding

X11 forwarding is an extension to TCP/IP forwarding. Because the authentication method to be used for the remote connections between X clients and servers is widely considered as vulnerable, secure connections for remote X clients and servers are strongly demanded. Figure 10-6 on page 243 illustrates a connection example between X clients and an X server. On svr02, the X client application process (Xclient2) connects to a local X server process. On svr01, another X client application process (Xclient1) connects to the remote X server process over the network.

To facilitate these connection topologies, the X protocol defines the DISPLAY environment value specified as host name:N.M:

- hostname - The host name where the X server process is running
- N - An integer value that defines a display number
- M - An integer value that defines a virtual display number

You use the integer value 0 for both N and M, unless you have multiple graphics displays.

In Figure 10-6, the local X client process (Xclient2) uses localhost:0.0 as the DISPLAY environment value and the remote X client process (Xclient1) uses svr02:0.0.

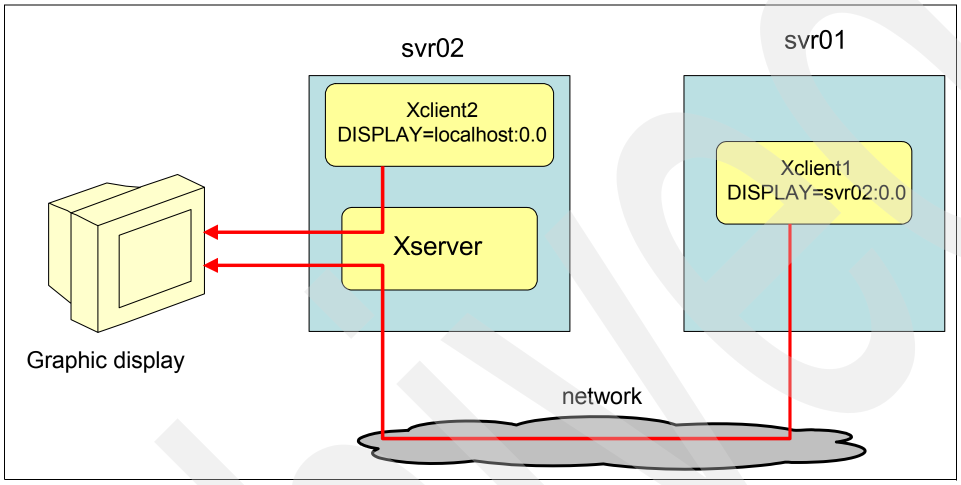


Figure 10-6 X11 forwarding

You can create a secure tunnel that protects remote connections between the X client and server processes over the network by using X11 forwarding. To use X11 forwarding, do the following:

1. Verify whether the remote OpenSSH server you are going to connect to supports X11 forwarding. If the /etc/ssh/sshd_config file on the remote server has the line shown in the following example, you have to change the no string to yes and restart the sshd process:

   ```
   X11forwarding no
   ```

2. On svr02, verify whether the local X server process grants remote connections using the **xhost** command. If the command returns the following output:

   ```
   emilia@svr02 $ xhost
   access control enabled, only authorized clients can connect
   ```

   then you have to issue the following command to instruct the X server process to grant remote connections from svr01:

   ```
   emilia@svr02 $ xhost +svr01.itsc.austin.ibm.com
   svr01.itsc.austin.ibm.com being added to access control list
   ```

3. On svr02, execute the following commands:

   ```
   emilia@svr02 $ ssh -l valen -X svr01
   ```

   The -X option instructs the ssh command to enable X11 forwarding. The command returns the command prompt on svr01, as shown in the following example:

   ```
   valen@svr02 $
   ```

4. Verify the DISPLAY environment value on the remote server:

```
valen@svr01 $ echo $DISPLAY
svr01.itsc.austin.ibm.com:10.0
```

Now, if you invoke X client applications, such as xclock and Netscape Navigator, in this session, those clients connect to the local X server process on svr02.

10.2.3 Positioning OpenSSH vs. SSL/TLS

Making OpenSSH available on z/OS answers the many requests received from UNIX System Services users to have an SSH-like facility.

As seen above, SSH was intended to provide a secure alternative to telnet and FTP by setting up an encrypted tunnel between hosts. Whereas SSL/TLS by itself does not provide any end user service, it actually is a way for programmers who are creating any kind of network applications that use TCP sockets to build *in the application* strong authentication, data confidentiality, and integrity for data exchange.

The two protocols are overlapping in their capabilities as SSH can also provide a secure tunnel to other applications via its TCP forwarding capability, the latter potentially extending the protection to a range of many TCP-based protocols.

It must also be noted that SSL/TLS exploits public keys packaged in x.509 certificates, and therefore requires using a Public Key Infrastructure (PKI), whereas OpenSSH accommodates raw key values that are not imbedded and certified in a digital certificate (although, strictly speaking, the IETF specifications open SSH to the use of digital certificates). In that respect OpenSSH requires a lighter infrastructure, as key certification is under the responsibility of a local administrator.

It must be noted that they are both using common algorithms, hence the OpenSSH implementation is using an imbedded OpenSSL code.

The z/OS UNIX telnet server does not provide SSL/TLS protection (TN3270 does). OpenSSH with its remote login function provides an equivalent capability. As it is also true that SSL-enabled FTP clients are not that common in the market, OpenSSH with sftp can provide an attractive alternative. Note, however, that today z/OS sftp does not provide access to MVS data sets.

10.3 z/OS OpenSSH implementation

In this section we describe the contents of the z/OS OpenSSH product. The reference document is *z/OS IBM Ported Tools for z/OS User's Guide*, SA22-7985.

10.3.1 Functions provided

OpenSSH now provides z/OS UNIX System Services users with the capability of using an OpenSSH client (ssh) and server (sshd) running on z/OS. It provides the following suite of connectivity tools:

- Secure remote login (ssh), an alternative to:
  - rlogin
  - rsh
- Secure file transfer (sftp and scp), an alternative to ftp and rcp.

Attention: The z/OS OpenSSH client cannot be invoked from the z/OS UNIX shell. This is working as designed, as there is no way to hide passwords entered in the shell.

Other basic utilities such as ssh-add, ssh-agent, ssh-keysign, ssh-keyscan, ssh-keygen, and sftp-server are also included.

The IBM Ported Tools for z/OS implementation of sshd support both SSH protocol Versions 1 and 2 simultaneously. The default sshd configuration runs only Protocol Version 2.

Functional content

In this section we discuss functional content.

OpenSSH client and server

Here we review the OpenSSH client and server.

- ssh: This is the OpenSSH client that establishes the secure channel with the OpenSSH server and performs the secure remote login program.
- sshd: Secure remote login daemon—a daemon that listens for connections from ssh clients and handles key exchange, encryption, authentication, command execution, and data exchange.

Together, ssh and sshd provide secure encrypted communications between two untrusted hosts over an insecure network. Once a SSH session is established, other connections can be forwarded over this secure channel, like X11, and other TCP-based protocols.

Secure file transfer

For secure file transfer:

- sftp (secure file transfer program) - An interactive file transfer program, similar to the ftp user interface. It performs all operations over an encrypted ssh transport and may also use many features of ssh. sftp requests flow through the OpenSSH client and requires the sftp-server (SFTP server subsystem) be operating on the server-side of the SFTP protocol. The sftp-server is automatically invoked from sshd.
- scp - Secure copy (remote file copy program) - It also uses the OpenSSH client and server for data transfer. The command syntax is similar to rcp; however, unlike rcp, scp asks for passwords/passphrases if necessary.

Key management

The utilities below are provided for key generation and management. Note that the code involved is specific to z/OS OpenSSH. There is no re-use of other key management code provided by IBM, such as GSKKYMAN, IKEYMAN, etc.

- ssh-keygen: Creates public/private key pairs.
- ssh-agent: An authentication agent that holds private keys in memory, saving you from retyping your passphrase repeatedly.
- ssh-add: Loads private keys into the agent memory.
- ssh-keyscan: Gathers SSH public host keys from the client, via an SSH protected conversation. This is used in preparation of server authentication by the client.

Helper applications

Some helper applications are:

- ssh-rand-helper: Entropy gathering code for random number generation

- ssh-askpass: X11 GUI for passphrase entry, called by ssh-add
- ssh-keysign: Helper program that performs the digital signatures required for host-based authentication

Prerequisites

OpenSSH for z/OS requires to be at z/OS 1.4 or later, with the PTFs shown in Table 10-1.

Table 10-1 z/OS OpenSSH required services levels

	z/OS 1.4	**z/OS 1.5**
Communications Server	APAR: PQ84183 PTF UQ85501	APAR: PQ82209 PTF: UQ83463
Communications Server	APAR: PQ80471 PTF: UQ82236	APAR: PQ80471 PTF: UQ82237
RSM	APAR: OA05893 PTF: UA09169	APAR: OA05893 PTF: UA09170

10.4 z/OS OpenSSH principles of operation

In this section we review z/OS OpenSSH principles of operation.

10.4.1 OpenSSH configuration files

There are many files involved in operating OpenSSH. The reference book lists all of them by categories such as program-generated files, administrator-generated files, and user-generated files. In this section we review the configuration files, both for the client and the server side, and the files used for authentication, as we describe the authentication mechanisms.

Note that some files that contain parameters intended to be used at the system level can be replaced by user-specific files located at $HOME (that is, the user's home directory). These user files contain sets of parameters that override the system files.

Server and client configuration files

Server and client configuration files are:

- /etc/ssh/sshd_config file - To control the SSH server's authentication methods allowed, protocols, and ciphers supported, port forwarding, and session control options.
- /etc/ssh/ssh_config file - To control the SSH-client-side authentication methods, protocols, ciphers, port forwarding settings, and session control option. Note that you may have a particular ssh_config per user if it is located in the user's $HOME/.ssh/config file; it then overrides the system-wide configuration file.

10.4.2 Server authentication

The server's authentication mechanism has been described at a high level in 10.2.1, "OpenSSH principles of operation" on page 238. Note that after the first connection, the $HOME/.ssh/known_hosts file is created on the client, if not created already. The file contains the connected server's public host key and is automatically updated with new server's public key. Therefore, you will not be prompted to confirm the server's fingerprint from the second connection attempt. Refer to Figure 10-7 on page 247 for a graphical representation of the exchange.

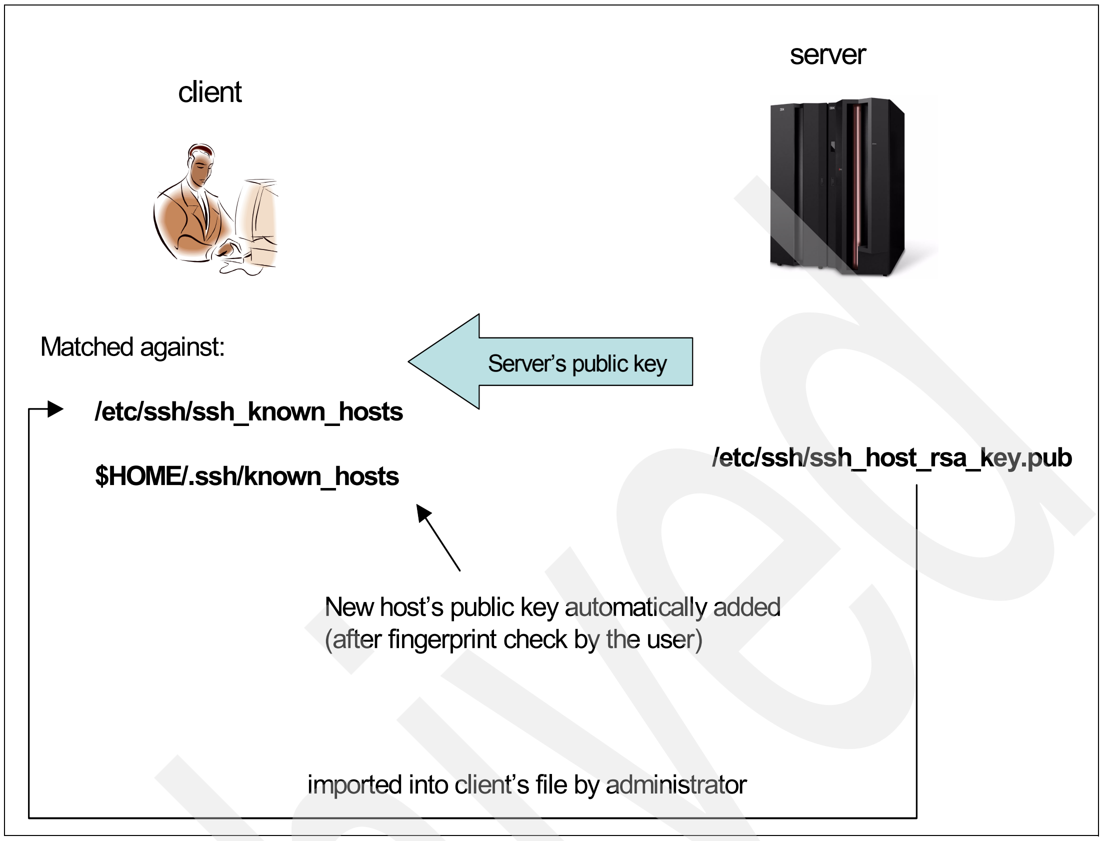

server
client
Matched against:
Server's public key
/etc/ssh/ssh_known_hosts
$HOME/.ssh/known_hosts
/etc/ssh/ssh_host_rsa_key.pub
New host's public key automatically added
(after fingerprint check by the user)
imported into client's file by administrator

Figure 10-7 OpenSSH server authentication

10.4.3 Client authentication

The possible client authentication methods to try are specified in the client configuration file with the PreferredAuthentication keyword. For z/OS they are:

- Host based
- Public key
- Password

Corresponding keywords in the server's configuration file must allow the client-selected methods.

Important: Remember that, at client authentication time, a secure channel has already been established between the OpenSSH client and server.

Host-based authentication

This form of authentication uses the traditional UNIX.rhosts or the specific SSH .shosts files in the remote server directory. These files contain lines of the form <client_system_name> <userID>, and the client is considered authenticated if there is a match between the current user ID and client system name (expected to be in a DNS name format), and one of the lines in these files. No password is required form the client but, as expected, it is a very insecure authentication mechanism. Note that .shosts file content is exploited only when accessing via OpenSSH, as opposed to .rhosts, which is intended to be used for regular accesses. Implementing .shosts only is a way to impose OpenSSH use.

Another option is to combine the .rhosts/.shosts authentication with the client host's public key; that is, the client user ID is signed by the client host's private key and the signature is verified at the server before looking into the .rhosts and .shosts files. This option is selected at in the client ssh.config file with the HostBasedAuthentication keyword, and is shown in Figure 10-8. The server configuration file must allow this authentication method via the HostBasedAuthentication keyword.

Attention: The $HOME directory on the server side is the home directory allocated to the client in the server. It is accessed on the basis of the identity passed by the client.

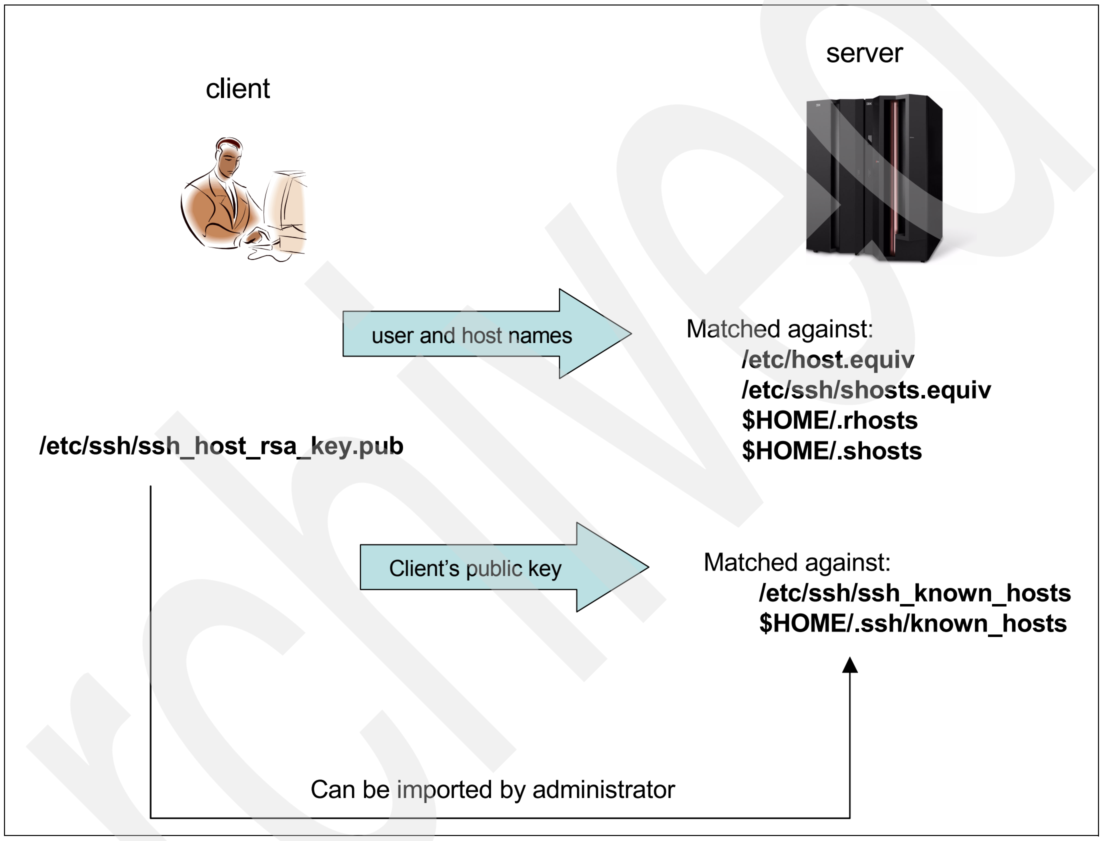


Figure 10-8 OpenSSH client host-based authentication

User public key

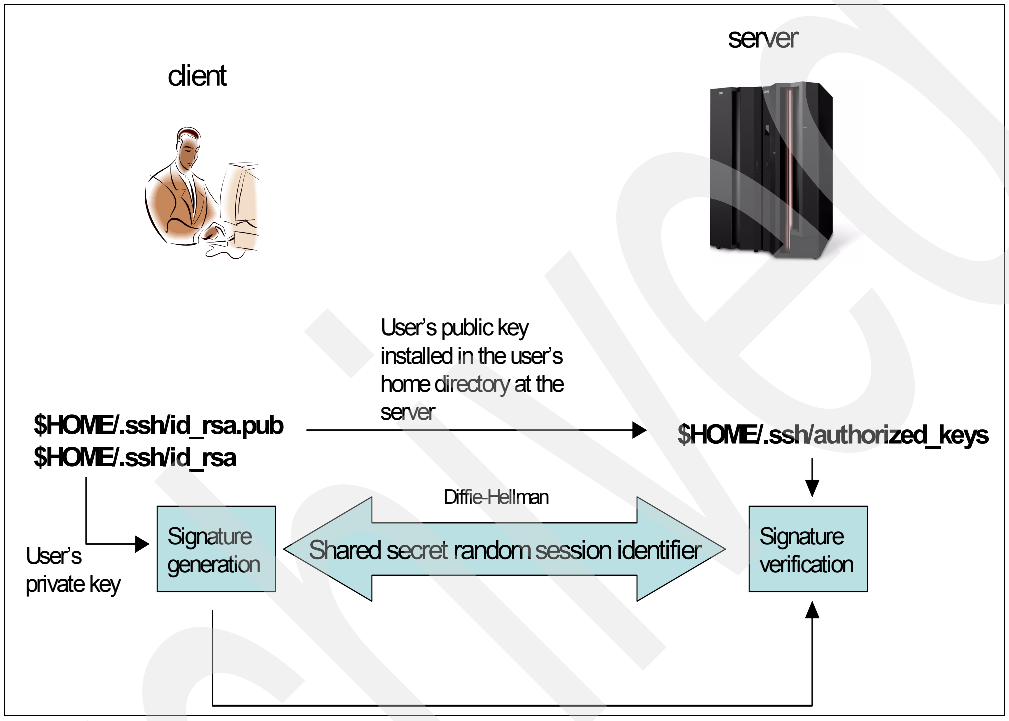
When this authentication mechanism is selected, the client authenticates using public key cryptography. The command ssh-keygen can be used to create public and private keys pairs. The key pair can be either RSA or DSA keys with the SSH2 protocol.

The private key is securely kept at the client, optionally encrypted with a passphrase, while the public key is installed in the server in the $HOME/.ssh/authorized_keys file. Proper client authentication is achieved by signing a secret random session number exchanged via the Diffie-Hellman algorithm, as shown in Figure 10-9 on page 249.

Note: It is expected that the user's private key be protected by a passphrase and the user to be prompted, on the client side, to enter this passphrase. This prompt can be avoided if a ssh-agent is operating on the client side.

This authentication method has to be permitted in the server's configuration file with the PubkeyAuthentication keyword.

Root is treated as any other user, with its files in the directory /.ssh. For root and other system logins you may want to use an empty passphrase when creating the key. This is especially true if you want to run cron jobs between machines as this user, because there will not be anyone there to provide the passphrase when the job runs. The passphrase does provide an additional level of security. Should someone break into your system the private key could be stolen, but without the passphrase they would not be able to exploit it on the remote system.



server
client
User's public key installed in the user's home directory at the server
$HOME/.ssh/id_rsa.pub
$HOME/.ssh/id_rsa
$HOME/.ssh/authorized_keys
Diffie-Hellman
User's private key
Signature generation
Shared secret random session identifier
Signature verification

Figure 10-9 OpenSSH user public key-based authentication

Password

When this authentication method is selected, a UNIX login password is sent from the client to the server via the encrypted tunnel. The z/OS OpenSSH server invokes SAF to verify the client user ID and password.

This authentication method has to be permitted in the server's configuration file with the PasswordAuthentication keyword.

10.4.4 z/OS OpenSSH and the syslogd daemon

The z/OS OpenSSH daemon (sshd) provides messages that can be written to syslogd. The log facilities it uses are:

```
DAEMON, USER, AUTH, LOCAL0, LOCAL1, LOCAL2, LOCAL3, LOCAL4, LOCAL5, LOCAL6, LOCAL7
```

The default is AUTH.

The log levels are:

```
QUIET, FATAL, ERROR, INFO, VERBOSE, DEBUG, DEBUG1, DEBUG2 and DEBUG3.
```

The default is INFO.

10.4.5 z/OS OpenSSH restrictions

The following user authentication methods are not supported on z/OS:

- Keyboard interactive
- Smart card device (The smart card is intended here to store the user's private key.)
- Kerberos tickets and AFS® tokens
- Challenge-response authentication

The following files are not supported in z/OS OpenSSH:

- /etc/hosts.allow, /etc/hosts.deny - These files are intended to enforce access controls by tcp-wrappers.

The following functions cannot be specified at the same time:

- Privilege separation and compression

z/OS being an EBCDIC platform, the following restrictions apply:

- User-defined subsystems are only supported when both the OpenSSH client and server are running on z/OS. This is due to a limitation in the SECSH protocol with regards to EBCDIC platforms. User-defined subsystems are specified by using the sshd_config subsystem keyword. Only the built-in sftp subsystem is supported for transfers between all platforms.
- OpenSSH does not run in multibyte locales.

10.4.6 Supported cryptographic algorithms

As of the writing of this redbook, all encryption algorithms are performed through software only, via statically linked OpenSSL routines, as opposed to using the z/OS hardware cryptography. The algorithms provided are:

- Data privacy - Symmetric encryption for session data
  - Advanced Encryption Standard (AES) with 128, 192, or 256 bit keys
  - arcfour
  - blowfish
  - DES, triple-DES
  - CAST-128

  The default order of preference, as specified in the ssh.config and sshd.config files, is:

  ```
  aes128-cbc,3des-cbc,blowfish-cbc,cast128-cbc,arcfour,aes192-cbc,aes256-cbc
  ```
- Data integrity - Hash algorithms
  - SSH Protocol Version 2 uses MD5, SHA-1, RIPEMD-160.
  - SSH Protocol Version 1 uses the weaker CRC-32.
- User and host authentication - Public key algorithms
  - RSA
  - DSA
- Session key exchange
  - Diffie-Hellman

10.5 Installing OpenSSH on z/OS

In this section we discuss installing OpenSSH on z/OS.

10.5.1 Steps for migrating from downloaded versions

Since there are many different sources where you might have downloaded a version of OpenSSH, use the **find** or **whence** commands to determine if any of the following programs exist on your system:

- ssh
- sshd
- scp
- sftp
- sftp-server
- ssh-add
- ssh-agent
- ssh-keygen
- ssh-keyscan
- ssh-keysign
- ssh-rand-helper
- ssh-askpass

Remove these programs or move them to a backup directory. Now you can continue with the installation of OpenSSH provided in the IBM Ported Tools for z/OS.

10.5.2 Ordering and install

Were able to order OpenSSH as part of IBM Ported Tools for z/OS, which is a non-priced product. This can be ordered from the following Web site:

https://www14.software.ibm.com/webapp/ShopzSeries/ShopzSeries.jsp

Once we received the OpenSSH, we then used SMP/E to complete the OpenSSH install. Data set JON.OPENSSH.SMPMCS was used as input to our SMP/E jobs.

In the receive job we entered JON.OPENSSH.SMPMCS in the SMPPTFIN dddef and coded the following RECEIVE parameters (Figure 10-10).

```
SET BDY(GLOBAL) .
RECEIVE SYSMODS
        SOURCEID(OPENSSH)
        RFPREFIX(JON.OPENSSH) .
```

Figure 10-10 Receive parameters

Once the receive job has run successfully, it produces two sample jobs, which we need to run. These jobs are independent of each other, so they can be run in any order. First we ran job FOTISDDF to define the SMP/E DDDEFs. This included the path name for which the OpenSSH will use, PATH('/usr/lib/ssh/IBM/'). The second job FOTISMKD invokes the supplied FOTMKDIR EXEC to allocate HFS paths. The FOTMKDIR is a rexx job, which besides creating new directories also adds files to existing directories.

Note: The FOTMKDIR REXX exec failed when we ran it. It actually had a missing end statement. Ensure that the IF $RC <>12 THEN DO has an END coded before you run the job, otherwise it will fail.

Now that we have run and created the PATHs, HFS directories, and files, we can now perform SMP/E APPLY and ACCEPT jobs. This completes the SMP/E install.

Install considerations

Ensure that you have CONTROL access to the SMP/E CSI data sets and READ access to the FACILITY class profile BPX.FILEATTR.PROGCTL, otherwise the jobs will fail with RACF access.

You must make sure that certain directories were set up correctly when z/OS UNIX was installed.

Table 10-2 Directories

Directory	Permission	Owner	Notes
/var/empty	755	UID(0)	Must be empty. It is used as the home directory for the SSHD (unprivileged) user.
/var/run	755	UID(0)	Holds the sshd.pid file, which contains the process ID of the most recently started OpenSSH daemon. If another directory is preferred, the PidFile configuration option can be specified in the daemon's sshd_config file.
/etc/ssh	755	UID(0)	Holds the configuration files for ssh and sshd.

The full installation procedures are documented in Program Directory for IBM Ported Tools for z/OS V1.1.0 Program Number 5655-M23 FMID HOS1110.

10.5.3 Setting up and starting the sshd daemon

Before we can start the sshd daemon, the configuration file must be created and edited, the setup for server authentication must be performed, and the sshd privilege separation user created. We are assuming that you managed to have a syslogd running in your system.

1. Copy the configuration files from the /samples directory to the /etc/ssh directory:
   - cp -p /samples/sshd_config /etc/ssh/sshd_config
   - cp -p /samples/ssh_config /etc/ssh/ssh_config
   - cp -p /samples/moduli /etc/ssh/moduli
   - cp -p /samples/ssh_prng_cmds /etc/ssh/ssh_prng_cmds

   All the permission bits should be 644.
2. Now we need to modify the /etc/ssh/sshd_config file to control the SSH server's authentication methods allowed, protocols, ciphers supported, port forwarding, and session control options. The only changes we made were to uncomment PORT, and we changed this to PORT 1024 and ListenAddress to the host IP address 10.11.12.13. Everything else we left to the defaults (Protocol Version 2 being the default).
3. We did not make any changes to the ssh_config file and left everything to default.
4. We had to amend our /etc/profile for the OpenSSH message catalog. The NLSPATH environment variable in the system-wide shell had to be changed to:

   ```
   NLSPATH=/usr/lib/nls/msg/%L/%N/usr/lib/nls/msg/%L/%N.cat
   ```
5. We need to set up for server authentication. Remember from the sshd_config file that we are using protocol 2, so we need to generate the host keys for the SSH server. Host keys

allow a client to verify the identity of the server. We used the ssh-keygen command, from the OMVS shell, to generate a host RSA key, as shown in Figure 10-11.

```
ssh-keygen -t rsa -f /etc/ssh/ssh_host_rsa_key -N ““

JON: /u/jon>ssh-keygen -t rsa -f /etc/ssh/ssh_host_rsa_key -N ""

Generating public/private rsa key pair.
Your identification has been saved in /etc/ssh/ssh_host_rsa_key.
Your public key has been saved in /etc/ssh/ssh_host_rsa_key.pub.
The key fingerprint is:
aa:a0:85:18:6f:25:de:0e:03:99:78:e3:2b:c9:31:47 JON@MOP08
```

Figure 10-11 Host keys

Before the first connection attempt using OpenSSH, the client has no idea about the remote server's host keys. If another host managed by a malicious user is masquerading as the target server, the client might not realize it. As an analogy, imagine you are calling a person for the first time. How can you trust that the voice belongs to the correct person if you have never heard his or her voice? To avoid this security risk, you should receive a host key fingerprint from the administrator who manages the server you are going to connect to before the first connection attempt. Upon the first connection attempt using OpenSSH, you will see the fingerprint of the remote server. You should verify whether this fingerprint matches the one received from the administrator.

6. We now need to create the sshd privilege separation user. Privilege separation, where the OpenSSH daemon creates an unprivileged child process to handle incoming network traffic, is enabled in the default configuration for sshd. You must assign a known unused group ID and unused nonzero user ID:

```
ADDGROUP SSHDG OW(OMVSGRP) SUPGROUP(OMVSGRP) OMVS(GID(9876)) +
ADDUSER SSHD DFLTGRP(SSHDG) OW(SSHDG) OMVS(UID(98765) +
HOME('/var/empty') PROGRAM('/bin/false')) NOPASSWORD
```

7. We now must define a user ID to the daemon. The user ID must have a UID of 0 and READ access to BPX.DAEMON. As we already had the user ID OEKERN setup on our system, we decided to use this ID. The sshd daemon is installed with the program control and nosharesas extended attributes. Ensure that the following are set in your PROGRAM * profile:

```
RALT PROGRAM * ADDMEM('CEE.SCEERUN'/'******'/NOPADCHK)
RALT PROGRAM * ADDMEM('SYS1.LINKLIB'/'******'/NOPADCHK)
```

8. Before we can start our sshd daemon as standalone, we need to create a proc in our system procedure library as described in Figure 10-12. This will invoke BPXBATCH, which sets up the standard file descriptors before running /SERVICE/usr/sbin/sshd.

```
//SSHD    PROC
//SSHD    EXEC PGM=BPXBATCH,REGION=0M,TIME=NOLIMIT,
//          PARM='PGM /SERVICE/usr/sbin/sshd -f /etc/ssh/sshd_config'
//* STDIN AND STDOUT ARE BOTH DEFAULTED TO /DEV/NULL
//STDERR    DD PATH='/tmp/sshd.stderr',
//             PATHOPTS=(OWRONLY,OCREAT,OAPPEND),PATHMODE=(SIRWXU)
```

Figure 10-12 SSHD Proc

9. As we have now set up a procedure for SSHD, we need to create the RACF started task. The user ID associated with this started task must have superuser and daemon authority; again we used OEKERN:

```
RDEFINE STARTED SSHD.* OW(OMVSGRP) UACC(NONE) STDATA(USER(OEKERN) TRUSTED(NO))
```

10. Now we can actually start the sshd daemon via SDSF; we simply type in `S SSHD`. Looking in the syslogd output file we can see the message shown in Figure 10-13.

```
sshd[83886724]: Server listening on 9.100.193.71 port 1024.
```

Figure 10-13 Output from sshd daemon in the syslog.log file

A TCP/IP Netstat command at the z/OS system console yields to display all existing connection yields. The results are shown in Figure 10-14.

```
D TCPIP,,N,ALLC
EZZ2500I NETSTAT CS V1R6 TCPIP 459
USER ID  CONN     LOCAL SOCKET           FOREIGN SOCKET         STATE
BPXOINIT 00000012 0.0.0.0..10007         0.0.0.0..0             LISTEN
FTPD1    00000014 0.0.0.0..21            0.0.0.0..0             LISTEN
LDAPSRV  0000001C 0.0.0.0..389           0.0.0.0..0             LISTEN
LDAPSRV  00000C84 10.11.12.13.389 10.21.22.21.62..49494    ESTBLSH
SSHD2    00002BAC 10.11.12.13..1024    0.0.0.0..0             LISTEN
TCPIP    00000010 127.0.0.1..1024        127.0.0.1..1025        ESTBLSH
TCPIP    0000000C 127.0.0.1..1024        0.0.0.0..0             LISTEN
TCPIP    0000000F 127.0.0.1..1025        127.0.0.1..1024        ESTBLSH
TN3270   00000013 0.0.0.0..23            0.0.0.0..0             LISTEN
TN3270   00002B35 9.10.11.12.13..23      10.31.32.33..1158      ESTBLSH
```

Figure 10-14 Netstat All Connections display 1

For more on setting up and starting the sshd daemon, the *z/OS IBM Ported Tools for z/OS User's Guide*, SA22-7985, documents everything that is needed.

10.6 Using OpenSSH on z/OS

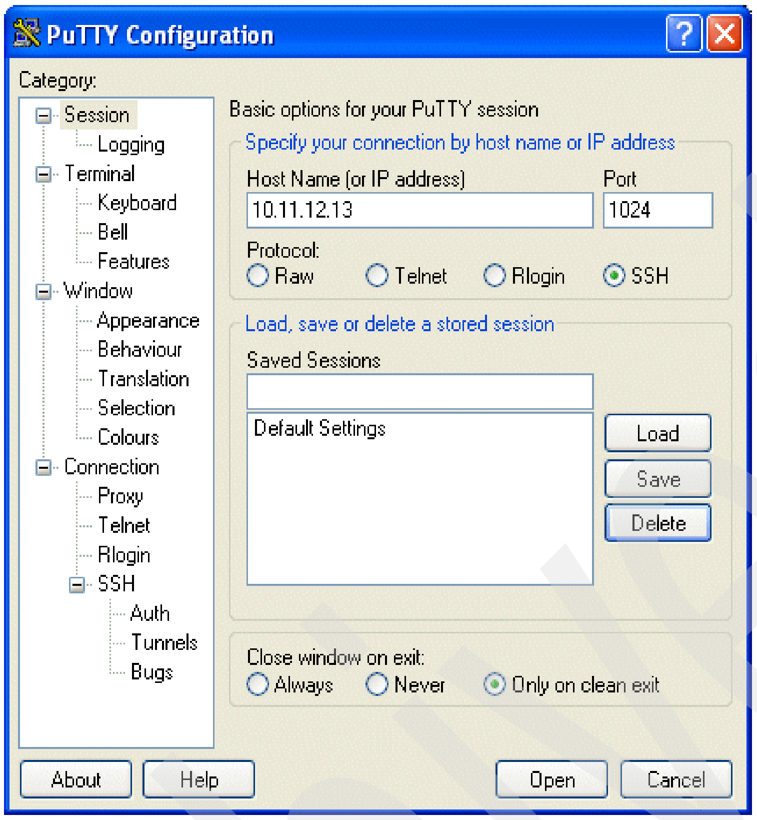
In this section we discuss using OpenSSH on z/OS.

10.6.1 Using PuTTY with password authentication

In this example we used PuTTY to verify that our OpenSSH works. PuTTY is a free SSH, telnet, and rlogin for 32-bit Windows systems. For detailed information about PuTTY, please visit the following URL:

http://www.chiark.greenend.org.uk/~sgtatham/putty

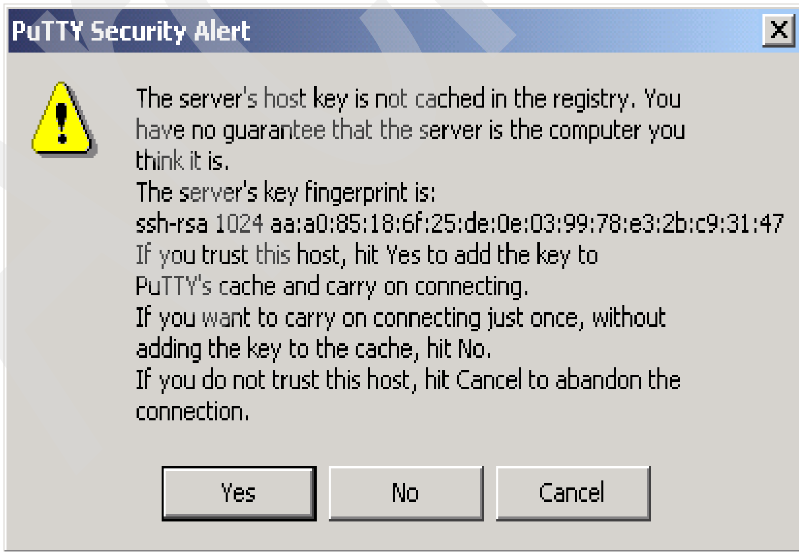
We installed PuTTY on our workstation, then executed the PuTTY and filled in the fields for our host and port, as shown in Figure 10-15 on page 255.



Host Name (or IP address): 10.11.12.13
Port: 1024
Protocol: Raw Telnet Rlogin [✓ SSH]
Saved Sessions:
Default Settings
Close window on exit: Always Never [✓ Only on clean exit]

Figure 10-15 PuTTY Configuration screen

Once we have indicated that we are using SSH and filled in our host name and port, click **Open** to gain access to our UNIX environment, where we will be prompted to enter our user ID and password. Before we gain access to the logon prompt, we received a message from PuTTY with a security alert; we know from the fingerprint that we are safe to continue.



The server's host key is not cached in the registry. You have no guarantee that the server is the computer you think it is.
The server's key fingerprint is:
ssh-rsa 1024 aa:a0:85:18:6f:25:de:0e:03:99:78:e3:2b:c9:31:47
If you trust this host, hit Yes to add the key to PuTTY's cache and carry on connecting.
If you want to carry on connecting just once, without adding the key to the cache, hit No.
If you do not trust this host, hit Cancel to abandon the connection.

Figure 10-16 PuTTY Security Alert

After entering the correct password, we are now in the z/OS UNIX shell, as shown in Figure 10-17 on page 256.

```
Login as: patkap
patkap@10.11.12.13's password:
PATKAP: /u/patkap:
```

Figure 10-17 Secure remote login with password authentication

Note that the syslogd daemon has stored the message shown in Figure 10-18, and a **netstat** command will show that another instance of sshd has started to serve the request from the authenticated user.

```
sshd[33555120]: Accepted password for patkap from 9.143.24.38 port 1106
```

Figure 10-18 Successful logon through OpenSSH

```
USER ID  CONN     LOCAL SOCKET           FOREIGN SOCKET         STATE
BPXOINIT 00000012 0.0.0.0..10007         0.0.0.0..0             LISTEN
FTPD1    00000014 0.0.0.0..21            0.0.0.0..0             LISTEN
LDAPSRV  0000001C 0.0.0.0..389           0.0.0.0..0             LISTEN
LDAPSRV  00000C84 9.100.192.109..389     9.100.194.62..49494    ESTBLSH
SSHD2    00002BFE 9.100.192.109..1024    9.143.24.85..1336      ESTBLSH
SSHD2    00002BF5 9.100.192.109..1024    0.0.0.0..0             LISTEN
TCPIP    00000010 127.0.0.1..1024        127.0.0.1..1025        ESTBLSH
TCPIP    0000000C 127.0.0.1..1024        0.0.0.0..0             LISTEN
TCPIP    0000000F 127.0.0.1..1025        127.0.0.1..1024        ESTBLSH
TN3270   00000013 0.0.0.0..23            0.0.0.0..0             LISTEN
TN3270   00002B35 9.100.192.109..23      9.143.24.85..1158      ESTBLSH
```

Figure 10-19 Netstat All Connections display 2

10.6.2 Using PuTTY with public key authentication

To use client public key authentication we need:

- To generate an RSA key pair at the PuTTY client, and export the public key to the z/OS server.
- To permit the SSH client and server to use and accept public key authentication. The method is actually authorized by default in the client PreferredAutehtication and in the server PubkeyAuthentication keywords in the configuration files.

Generation of the RSA key pair

To call the PuTTY key generator program we select the **PuTTY gen** shortcut that was created during PuTTY installation (alternatively we can invoke puttygen.exe). The PuTTY key generator window shown in Figure 10-20 appears.

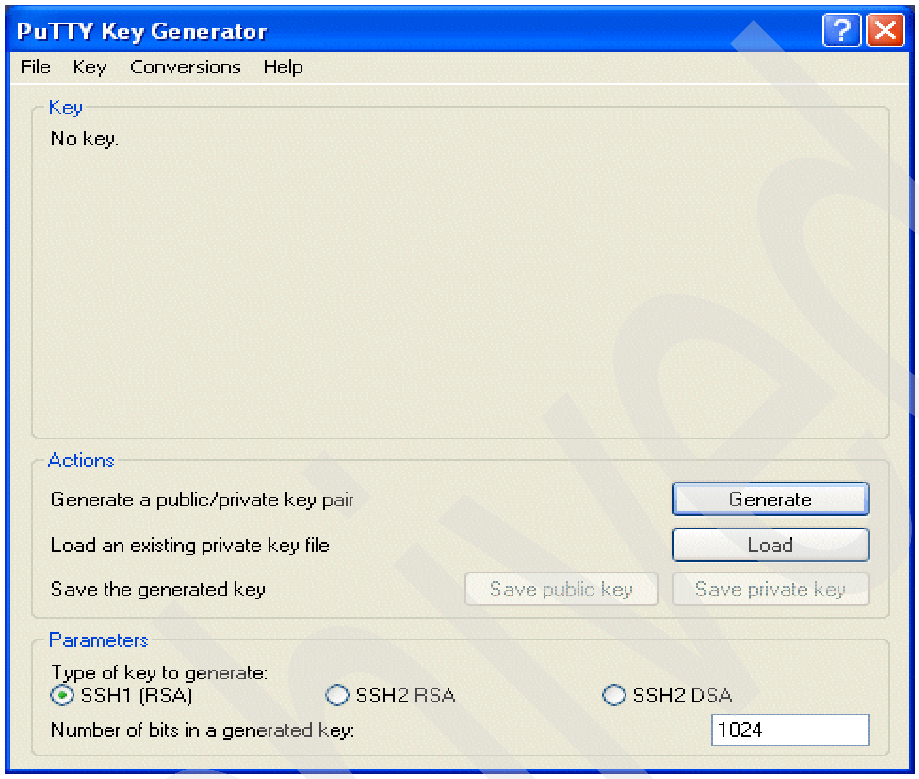
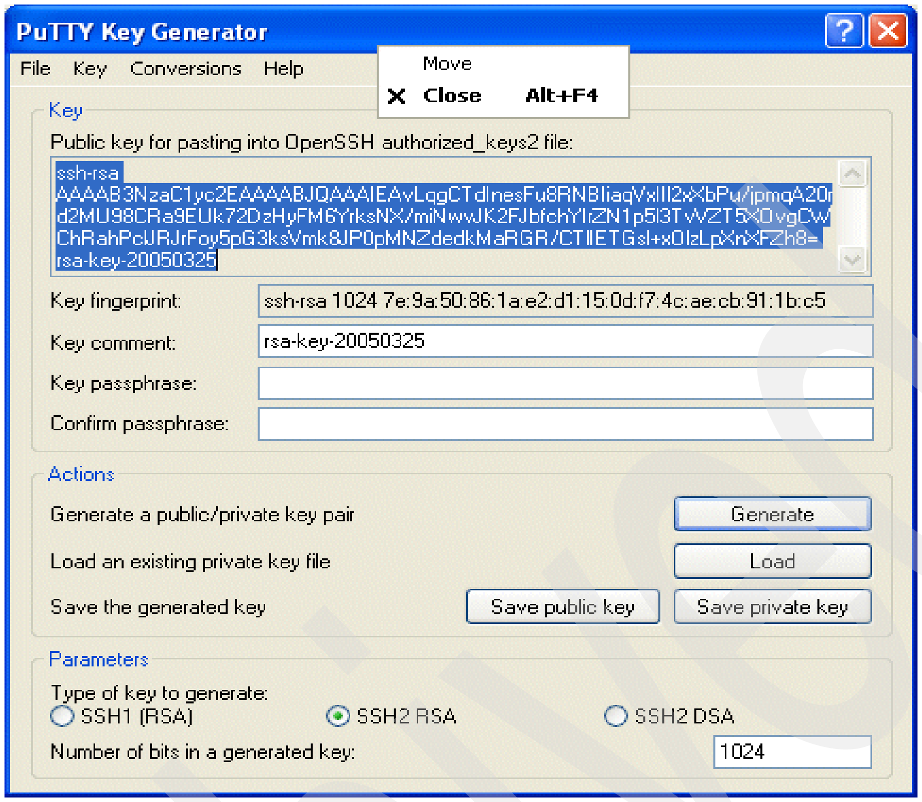


Figure 10-20 The PuTTY key generation window

We select the algorithm to be used (SSH2 RSA). The length of the key remains at 1024 bits, and we click the **Generate** button. During the generation process we are asked to move the cursor around to see the pseudo random number generator, and the final result appears as shown in Figure 10-21 on page 258. The public key is shown as a base-64 encoded value, meaning that it can be transferred by a copy and paste operation from terminal window to terminal window. There is also the option to save the public key in a file that can be FPTed (as ascii text to the server platform). The format of this key save file is shown in Figure 10-22 on page 258. The public key has to be installed in the end user $HOME/.ssh/authorized_keys file in the SSH server platform. On z/OS the key must be on a single line with the following format:

```
ssh-rsa base64_encoded_key_value
```

The private key must be saved protected by a passphrase. Enter your passphrase twice in the proper fields and click **Save private key**. You are then directed to choose the path and file name for your PuTTY private key file.



Public key for pasting into OpenSSH authorized_keys2 file:

```
ssh-rsa
AAAAB3NzaC1yc2EAAAABJQAAAIEAvLqgCTdlnesFu8RNBIiaqVxlIl2xXbPu/jpmqA20r
d2MU98CRa9EUk72DzHyFM6YrksNX/miNwwJK2FJbfchYIrZN1p5l3TvVZT5X0vgCW
ChRahPcURJrFoy5pG3ksVmk8JP0pMNZdedkMaRGR/CTIlETGsl+x0IzLpXnXFZh8=
rsa-key-20050325
```

Key fingerprint: ssh-rsa 1024 7e:9a:50:86:1a:e2:d1:15:0d:f7:4c:ae:cb:91:1b:c5

Key comment: rsa-key-20050325

Key passphrase:

Confirm passphrase:

Generate a public/private key pair — Generate

Load an existing private key file — Load

Save the generated key — Save public key — Save private key

Type of key to generate: SSH1 (RSA) [✓ SSH2 RSA] SSH2 DSA

Number of bits in a generated key: 1024

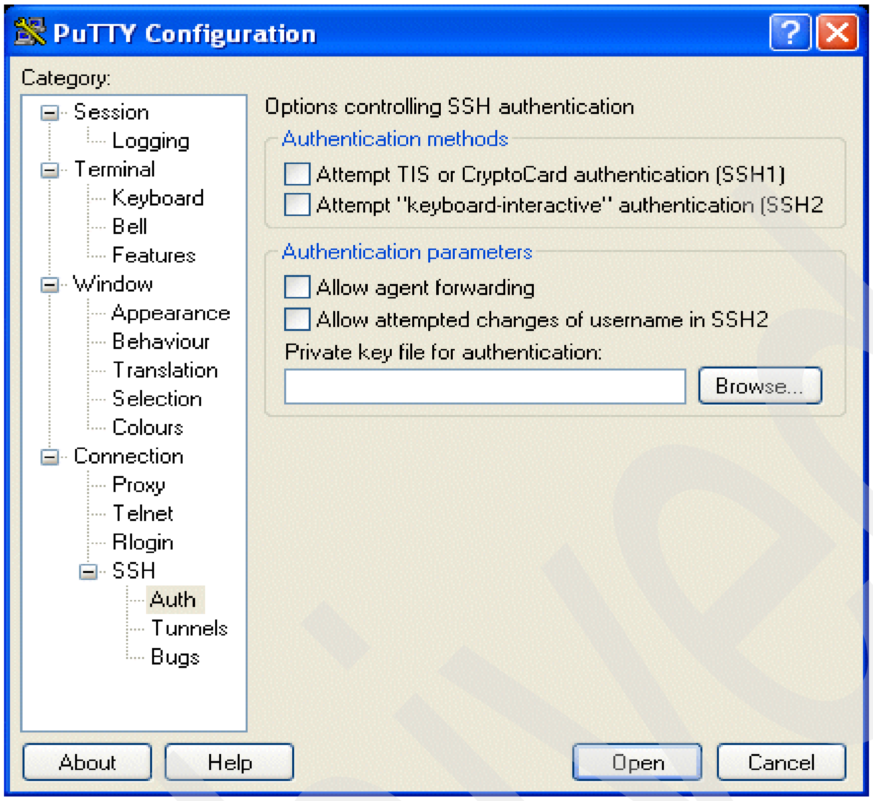
Figure 10-21 PuTTY and completed key generation

```
---- BEGIN SSH2 PUBLIC KEY ----
Comment: "rsa-key-20050418"
AAAAB3NzaC1yc2EAAAABJQAAAIBx10L4MUBs/zQPfbWZ+kvWQ9jY2DyVkqeed+tx
w87mWNZdxhXa2+oVTvaoiSgsQ0CSxkF24oF5IBsPCqwu1DSsz0Sou5SDX066rhrl
6XUgbtQwWAPVvIswBK/pCJzm4oKS5AFGGTQth0IsY5ug26Hm2jeOMpCI184opeMd
25W8dw==
---- END SSH2 PUBLIC KEY ----
```

Figure 10-22 Public key save file in PuTTY

Performing public key authentication

The PuTTY client must be reloaded with your normal session attributes. Then select the SSH Auth option category, as shown in Figure 10-23 on page 259. Retrieve your PuTTY private key file and click the **Open** button. The communication with the SSH server is initiated, and at client authentication time you will be asked on the login screen, after entering your user ID, to enter the private key passphrase. Once entered correctly, the login sequence proceeds with public key authentication. When successful, you again have access to the z/OS UNIX shell, as shown in Figure 10-24 on page 259.



Options controlling SSH authentication

Authentication methods

- [ ] Attempt TIS or CryptoCard authentication (SSH1)
- [ ] Attempt "keyboard-interactive" authentication (SSH2

Authentication parameters

- [ ] Allow agent forwarding
- [ ] Allow attempted changes of username in SSH2

Private key file for authentication:

Figure 10-23 Preparing for client public key authentication

```
Login as: jon
Authenticating with public key “rsa-key-20050325”
Passphrase for key “rsa-key-2005325”:
JON: /u/jon:
```

Figure 10-24 PuTTY login with public key authentication

Using PuTTY for TCP port forwarding

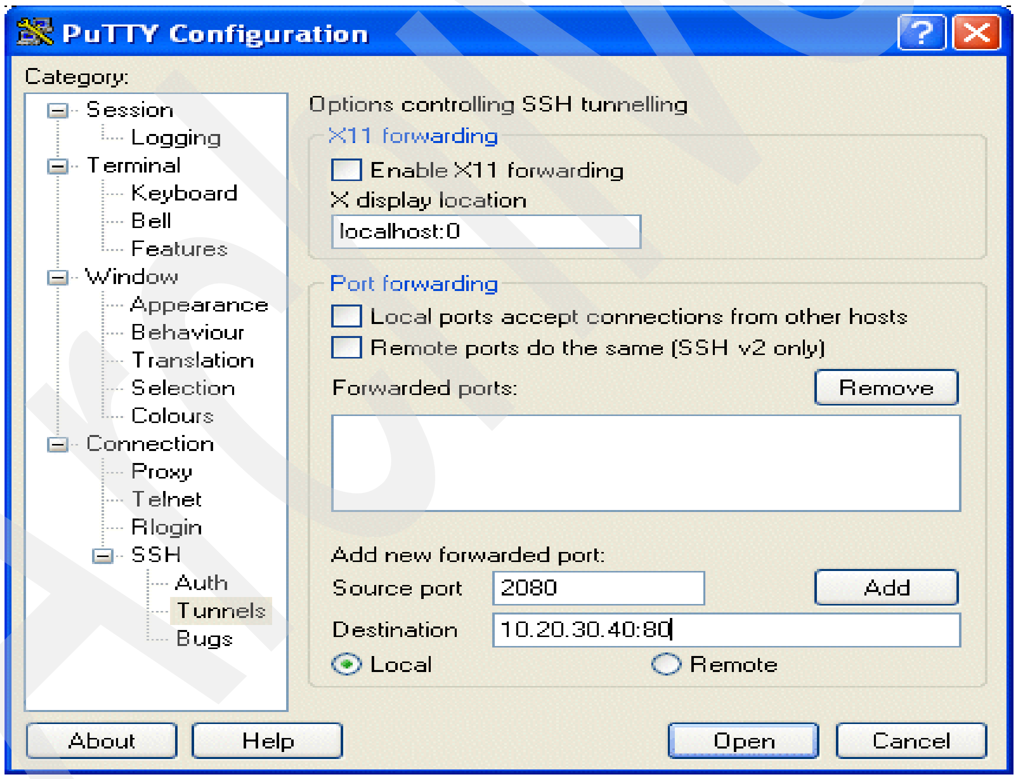
We are indicating here how to set up PuTTY so that you can flow the http protocol inside the OpenSSH tunnel, using the port forwarding function.

To establish TCP port forwarding first reload in PuTTY your session parameters. Then select **Tunnels** in the SSH options category. In the new window you can then specify a local or remote port forwarding, with the forwarded port number and the target system address and port. In the example shown in Figure 10-25 and Figure 10-26 on page 261, the local forwarded port 2080 is ending up addressing IP address 10.20.30.40, port 8000, from the SSH server side.

Once the forwarded port has been specified, you can come back to the login session, do a save of your session parameters (this will save the port forwarding information), and open the session with the server using the proper authentication method. Once client authentication has been performed and the secure tunnel has been established, you can start a browser session on your PuTTY client workstation, indicating the local host and the forwarded port in the URL that is here:

```
http://localhost:2080
```

The browser opens a connection, as shown in Figure 10-27 on page 261. This is received by the PuTTY SSH client, which has been set up to forward any local connection done to port 2080 to the SSH server. In turn, the SSH server has been told by the protocol to carry this request over to the host at IP address 10.20.30.20 to the service on port 80, which happens to be the target http server. Hence the http connection is protected by the SSH secure tunnel between the PuTTY client and the SSH server.



Options controlling SSH tunnelling

X11 forwarding

[ ] Enable X11 forwarding

X display location: localhost:0

Port forwarding

[ ] Local ports accept connections from other hosts

[ ] Remote ports do the same (SSH v2 only)

Forwarded ports:

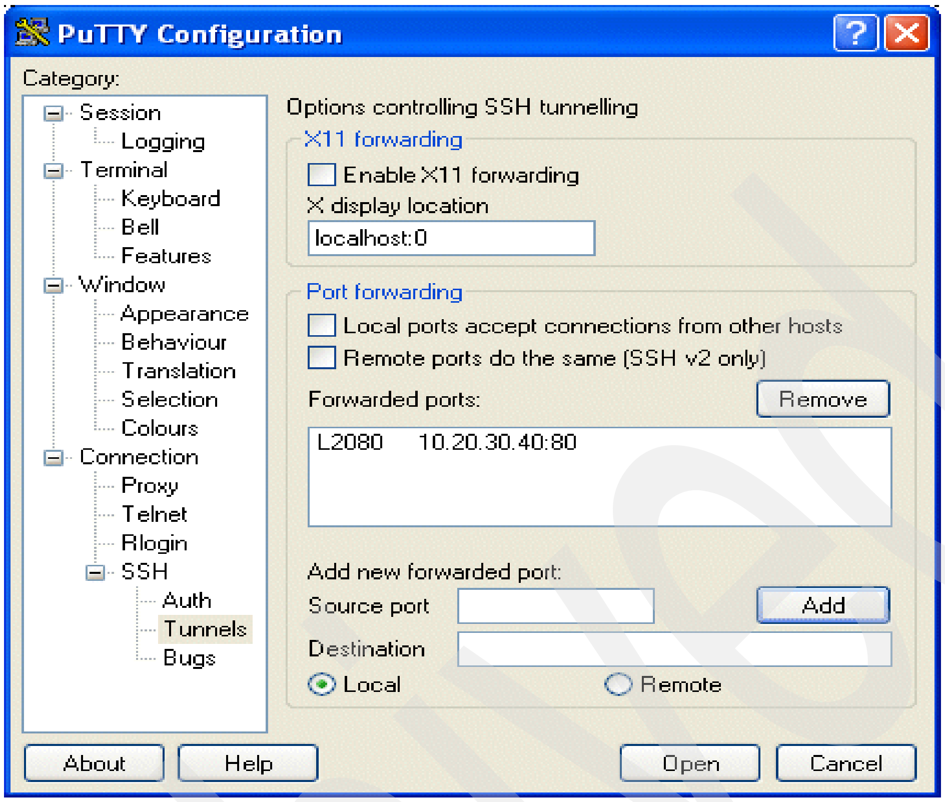
Add new forwarded port:

Source port: 2080

Destination: 10.20.30.40:80

[✓ Local] Remote

Figure 10-25 PuTTY port forwarding setup 1/2



Options controlling SSH tunnelling

X11 forwarding

[ ] Enable X11 forwarding

X display location: localhost:0

Port forwarding

[ ] Local ports accept connections from other hosts

[ ] Remote ports do the same (SSH v2 only)

Forwarded ports:

```
L2080     10.20.30.40:80
```

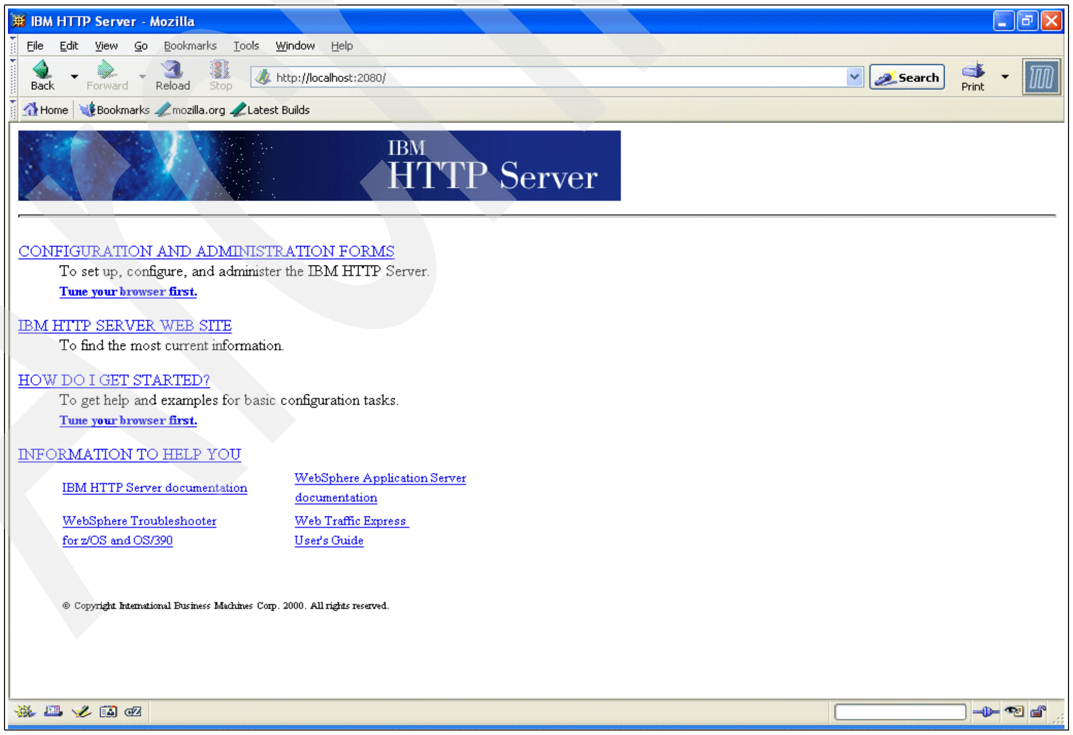
Add new forwarded port:

Source port:

Destination:

[✓ Local] Remote

Figure 10-26   PuTTY port forwarding setup 2/2


IBM HTTP Server - Mozilla
File Edit View Go Bookmarks Tools Window Help
Back Forward Reload Stop
Search
Print
Home Bookmarks mozilla.org Latest Builds
IBM
HTTP Server

http://localhost:2080/

CONFIGURATION AND ADMINISTRATION FORMS
To set up, configure, and administer the IBM HTTP Server.
Tune your browser first.

IBM HTTP SERVER WEB SITE
To find the most current information.

HOW DO I GET STARTED?
To get help and examples for basic configuration tasks.
Tune your browser first.

INFORMATION TO HELP YOU

IBM HTTP Server documentation — WebSphere Application Server documentation

WebSphere Troubleshooter for z/OS and OS/390 — Web Traffic Express User's Guide

© Copyright International Business Machines Corp. 2000. All rights reserved.

Figure 10-27   http port forwarding

A

EIM API demo sample code

List of the EIM APIs as of the writing of this book

This is the list refered to in “The EIM APIs” on page 167.

```
eimAddAccess
eimAddApplicationRegistry
eimAddAssociation
eimAddIdentifier
eimAddPolicyAssociation   (new at z/OS 1.6)
eimAddPolicyFilter             (new at z/OS 1.6)
eimAddSystemRegistry
eimChangeDomain
eimChangeIdentifier
eimChangeRegistry
eimChangeRegistryAlias
eimChangeRegistryUser
eimConnect
eimConnectToMaster
eimCreateDomain
eimCreateHandle
eimDeleteDomain
eimDestroyHandle
eimErrString
eimFormatPolicyFilter        (new at z/OS 1.6)
eimFormatUserIdentity        (new at z/OS 1.6)
eimGetAssociatedIdentifiers
eimGetAttribute
eimGetRegistryNameFromAlias
eimGetTargetFromIdentifier
eimGetTargetFromSource
eimGetVersion         (new at z/OS 1.6)
eimListAccess
eimListAssociations
eimListDomains
eimListIdentifiers
eimListPolicyFilters (new at z/OS 1.6)
eimListRegistries
eimListRegistryAliases
eimListRegistryAssociations (new at z/OS 1.6)
eimListRegistryUsers
eimListUserAccess
eimQueryAccess
eimRemoveAccess
eimRemoveAssociation
eimRemoveIdentifier
eimRemovePolicyAssociation (new at z/OS 1.6)
eimRemovePolicyFilter           (new at z/OS 1.6)
eimRemoveRegistry
eimRetrieveConfiguration        (new at z/OS 1.6)
eimSetAttribute
eimSetConfiguration               (not supported on z/OS)
eimSetConfigurationExt           (new at z/OS 1.6)
```

Figure A-1   List of EIM APIs, as of the writing of this book

Sample code to set up and run the EIM demo

This appendix provide setup scripts and the source code (C and html) for the EIM demo refred to in 7.4.1, “The eimdemo application” on page 177.

```
<HTML>
<HEAD>
<TITLE>EIM Demo</TITLE>
</HEAD>
<BODY BGCOLOR="WHEAT">
<H1 ALIGN="CENTER">EIM Demo</H1>
<HR WIDTH="60%" SIZE=6 NOSHADE>
<p>
This demo simulates a web application that access protected resources.
It uses EIM to obtain a user ID belonging to an employee and
accesses a resource procted by the user id.
<p>
Here's the scenario...
<ul>
<li>The help desk logs on to the application with their RACF
user ID and password
<li>The user fills in a field with an employee name or number
<li>The user clicks on the "Access Data" button.
</ul>
This action causes a CGI program to run.  The program:
<ul>
<li>Retrieves the employee name or number from the web page.
<li>Spawns a process to retrieve the local user ID for the
employee.
<li>Spawns a second process under the target user ID.  The process
opens the file /u/<i>target userid</i>/test.data, reads the
first line, and returns the content of the file.
<li>Builds a web page dynamically and returns it to the browser.
</ul>
<P>
<HR>
<form method=POST action="/demo-cgi-bin/eimdemo">
<P>
<PRE>
Enter the employee identifier <input type="text" name="employee"
size=40 maxlength=512 value="">
</PRE>
<HR>
<input type="hidden" name="webpage" value="1">
<P>
<PRE>
<CENTER>
<input type='submit' name='pushbutton' value='Access Data'><input type="submit"
name="pushbutton" value="Step by Step">
<HR>
</PRE>
</BODY>
</HTML>
```

Figure A-2 eimdemo.htm

Main program: eimdemo.c

```
/* Includes */
#undef  environ
#define _SHARE_EXT_VARS
#include <errno.h>
#include <spawn.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <eim.h>
#define _POSIX_SOURCE
#include <unistd.h>
#include <sys/wait.h>
#define TRUE 1
#define CHANGE 1

/* Definition of structure used in linked list of variables */
typedef struct _argument {
  char             *VariableName;
  char             *Value;
  struct _argument *pNext;
  } PARMLIST, *PPARMLIST;


/* Function definitions */
int         DisplayResults (char *, char *, char *, char *, char *,
                            char *, char *, char *);
int         DisplayResults2();
int         ErrMsg (char *msg, char *pValue);
int         EimErr (char *msg, int rc, EimRC *err,
                    char *, char *, char *, char *);
PPARMLIST   ReadArguments(int InputLength);
static void ASCIIToEBCDIC(char *Str);
static void PlusesToSpaces(char *Str);
static int  HexVal(char c);
static void TranslateEscapes(char *Str);
```

Figure A-3 eimdemo.c 1/10

```
/*******************************************************************/
/*                                                                 */
/* Function       : main                                           */
/*                                                                 */
/* Description    : This is a CGI program performs                 */
/*                  1) eimGetTargetFromIdentifier to retrieve the  */
/*                     local user id of an employee,               */
/*                  2) eimListIdentifiers to get general info      */
/*                     about the employee                          */
/*                  3) Spawns a program that reads a file owned    */
/*                     by the local user ID, and                   */
/*                  4) Returns a dynamic web page with information */
/*                     about the employee and the information from */
/*                     the file.                                   */
/*                                                                 */
/*******************************************************************/
main(int argc, char *argv[]) {
int  rc = 0;
  int  i  = 0;

  /* Environment variables of interest */
  char     *requestMethod = NULL;  /* REQUEST_METHOD               */
  char     *contentLength = NULL;  /* CONTENT_LENGTH               */
  char     *pBpxUserid    = NULL;  /* _BPX_USERID                  */

  /* Linked list of the field values taken from CONTENT_LENGTH */
  int       argLength = 0;
  PPARMLIST pParm     = NULL;
  PPARMLIST pHead     = NULL;

  /* EIM related variables                                     */
  char           eimerr[200];
  EimRC         *err           = (EimRC *)eimerr;
  EimHandle      handle;
  EimConnectInfo connInfo;

  char              *pDomainUrl        = NULL;

  char              *pEmployee         = NULL;
  EimIdentifierInfo  idName;
  char              *pTgtReg           = NULL;
  char               listData[4000];
  EimList           *pListTgtUserNames = NULL;
  EimTargetIdentity *pTgtEntry         = NULL;
  char              *pUserName         = NULL;
  char               tgtUserName[9];

  char               listData2[4000];
  EimList           *pListIdentifierInfo = NULL;
  EimIdentifier     *pIdentEntry         = NULL;
  char               uniqueName[100];
  char              *pUniqueName         = NULL;
  char               description[100];
  char              *pDescription        = NULL;
  char               aliasName[100];
  EimIdentifierName *pAliasNameEntry     = NULL;
  char              *pAliasName          = NULL;
```
Figure A-4   eimdemo.c 2/10

```
/* Parms required by __spawnp2                                */
  char                *file     = NULL;
  int                  fd_count = 0;
  int                  fd_array[3];
  struct __inheritance  inherit;
  char                *args[20];
  pid_t                pid;

  /* Setup program so reason codes are part of error messages */
  rc=setenv("_EDC_ADD_ERRNO2", "1", CHANGE);

  /* Get request method - only POST is recognized by this program */
  /* -- POST must be in uppercase (this avoids use of stricmp)    */
  requestMethod = getenv("REQUEST_METHOD");
  if ((requestMethod == NULL) || (strcmp(requestMethod, "POST"))) {
    ErrMsg("REQUEST_METHOD environment variable not properly set to
POST\n",requestMethod);
    return(0);
    }

  /* Get the fields from the web page and store them in a      */
  /* linked list.                                              */
  contentLength = getenv("CONTENT_LENGTH");
  if (contentLength == NULL) {
    ErrMsg("CONTENT_LENGTH environment variable not set\n",NULL);
    return(0);
    }
  argLength = atoi(contentLength);
  pHead = ReadArguments(argLength);
  pParm = pHead;

  /* Extract selected field values from the linked list        */
  while (pParm) {
    if (!strcmp(pParm->VariableName,"employee"))
      pEmployee = pParm->Value;

    pParm = pParm->pNext;
    }
```

Figure A-5 eimdemo.c 3/10

```
/* Get the employee's local user id from EIM.                */
  err->memoryProvidedByCaller = 200;
  pDomainUrl = NULL;                         /* get from RACF  */
  rc = eimCreateHandle(&handle,pDomainUrl,err);
  if (rc) {
   EimErr("eimCreateHandle failed",rc,err,pDomainUrl,pTgtReg,NULL,NULL);
   return(0);
   }

  connInfo.type = EIM_SIMPLE;
  connInfo.creds.simpleCreds.protect = EIM_PROTECT_NO;
  connInfo.creds.simpleCreds.bindDn  = NULL; /* get from RACF  */
  connInfo.creds.simpleCreds.bindPw  = NULL; /* get from RACF  */
  rc=eimConnect(&handle,connInfo,err);
  if (rc) {
    EimErr("eimConnect failed",rc,err,pDomainUrl,pTgtReg,
           connInfo.creds.simpleCreds.bindDn,
           connInfo.creds.simpleCreds.bindPw);
    return(0);
    }

  idName.id.name    = pEmployee;
  idName.idtype     = EIM_NAME;
  pTgtReg           = NULL;                  /* Get from RACF  */
  pListTgtUserNames = (EimList *)listData;
  rc=eimGetTargetFromIdentifier(&handle,&idName,pTgtReg,NULL,
                               4000,pListTgtUserNames,err);
  if (rc == 0) {
    pListIdentifierInfo = (EimList *)listData2;
    rc=eimListIdentifiers(&handle,&idName,4000,pListIdentifierInfo,err);
    if (rc) {
      EimErr("eimListIdentifiers failed",rc,err,pDomainUrl,pTgtReg,
             connInfo.creds.simpleCreds.bindDn,
             connInfo.creds.simpleCreds.bindPw);
      return(0);
      }
    }

  rc=eimDestroyHandle(&handle,err);

  /* convert the user name in the EimList structure to a string */
  bzero(&tgtUserName,sizeof(tgtUserName));
  if (pListTgtUserNames->entriesReturned > 0) {
    pTgtEntry = (EimTargetIdentity *)
             ((char *)pListTgtUserNames + pListTgtUserNames->firstEntry);
    pUserName      = (char *)pTgtEntry + pTgtEntry->userName.disp;
    strcpy(tgtUserName,pUserName);
    }/* convert the identifier info EimList structure to strings   */
```

Figure A-6   eimdemo.c 4/10

```
bzero(&uniqueName, sizeof(uniqueName));
bzero(&description, sizeof(description));
bzero(&aliasName, sizeof(aliasName));
if (pListIdentifierInfo->entriesReturned > 0) {
  pIdentEntry = (EimIdentifier *)
    ((char *)pListIdentifierInfo + pListIdentifierInfo->firstEntry);

  pUniqueName    = (char *)pIdentEntry +
                   (* pIdentEntry).uniquename.disp;
  strcpy(uniqueName,pUniqueName);

  pDescription   = (char *)pIdentEntry +
                   (* pIdentEntry).description.disp;
  strcpy(description,pDescription);

  pAliasNameEntry = (EimIdentifierName *)((char *)pIdentEntry +
                    (* pIdentEntry).names.disp);
  for (i=1; i<=(* pIdentEntry).names.listNum; i++) {
    pAliasName      = (char *)pAliasNameEntry +
                      (* pAliasNameEntry).name.disp;
    if (i > 1) strcat(aliasName,"; ");
    strcat(aliasName,pAliasName);
    pAliasNameEntry = (EimIdentifierName *)((char *)pAliasNameEntry +
                    (* pAliasNameEntry).nextEntry);
    }
  }

/* Get environment variable                                  */
pBpxUserid  = getenv("_BPX_USERID");

/* Write the first half of the HTML document.  The spawned   */
/* program will write the second half of the HTML document.  */
rc = DisplayResults(pEmployee,
                    pUniqueName, pDescription, aliasName,
                    pDomainUrl, pTgtReg, tgtUserName,
                    pBpxUserid);

if ((pUniqueName) && (tgtUserName[0])) { /* Employee found in EIM */
  /* Spawn a process under the target user id                 */
  file="/u/WEBSRV/testexec/eimdemo2";  /* program to run       */

  fd_count    = 3;                     /* file descriptor map  */
  fd_array[0] = SPAWN_FDCLOSED;        /* index - local fd     */
  fd_array[1] = 1;                     /* value - spawned pgm fd */
  fd_array[2] = 2;

  bzero(&inherit, sizeof(struct __inheritance)); /* new userid */
  inherit.flags = SPAWN_SETUSERID;
  strcpy(inherit.userid,tgtUserName);
  rc=setenv("_BPX_USERID",tgtUserName,CHANGE);

  args[0] = "/u/WEBSRV/testexec/eimdemo2"; /*  program to run  */
  args[1] = NULL;
  pid  = __spawnp2(file, fd_count, fd_array, &inherit,
                  (const char **) args, (const char **) environ);
  if (pid == -1) ErrMsg("__spawnp2 failed",strerror(errno));

} else { /* Either the identifier or the target user was not found */
  rc = DisplayResults2();
  }
}/* main */
```

Figure A-7   eimdemi.c 5/10

```
/******************************************************************************/
/*                                                                            */
/* Function       : ReadArguments                                             */
/*                                                                            */
/* Description    : Read the arguments from stdin that are supplied           */
/*                  to a CGI program when the method is POST.                 */
/*                  Breaks up the input into                                  */
/*                  (Variable, Value) pairs.                                  */
/*                  Handles translating of all the special characters         */
/*                  that HTTP puts into the strings.                          */
/*                                                                            */
/******************************************************************************/

PPARMLIST ReadArguments(int InputLength){
  PPARMLIST pCur  = NULL;
  PPARMLIST pHead = NULL;
  PPARMLIST pPrev = NULL;
  char     *pInput;
  char     *pToken;

  /* Read in the input                             */
  /* - ck for 1) no input parms, 2) malloc failure */
  if (InputLength < 1) {
    return(NULL);
    }
  pInput = malloc(InputLength + 1);
  if (pInput == NULL) {
    return(NULL);
    }
  gets(pInput);

  /* Variables are separated by the "&" character */
  pToken = strtok(pInput, "&");

  while (pToken) {
    /* Create and fill in linked list of variable information */
    pCur = malloc(sizeof(PARMLIST));
    pCur->VariableName = pToken;
    pToken = strchr(pToken, '=');
    if (pToken) {
      *pToken = '\0';           /* overwrite = with a NULL */
      pCur->Value = ++pToken;
      PlusesToSpaces( pToken );
      TranslateEscapes( pToken );
      ASCIIToEBCDIC( pToken );
      }
    else {
      pCur->Value = NULL;
      }
    if (pPrev) {
      pPrev->pNext = pCur;
      }

    if (!pHead) {
      pHead = pCur;
      }
    pPrev = pCur;
    pToken = strtok(NULL, "&");
    }

  if (pHead) {
    pPrev->pNext = NULL;
    }
  return(pHead);
  }/* ReadArguments */
```

Figure A-8 eimdemo.c 6/10

```
/*****************************************************************************/
/*                                                                           */
/* Function       : ASCIIToEBCDIC  (STATIC)                                  */
/*                                                                           */
/* Description    : This one's easy. It just translates any '+'              */
/*                                                                           */
/*****************************************************************************/

static void ASCIIToEBCDIC(char *Str) {
  if (Str != NULL) {
    while (*Str != '\0') {
      switch (*Str) {
        case 0x2F : *Str = '/'; break;
        default   :             break;
        }
      ++Str;
      }
    }
  } /* ASCIIToEBCDIC */

/*****************************************************************************/
/*                                                                           */
/* Function       : PlusesToSpaces (STATIC)                                  */
/*                                                                           */
/* Description    : This one's easy. It just translates any '+'              */
/*                  characters found into ' ' characters.                    */
/*                                                                           */
/*****************************************************************************/

static void PlusesToSpaces(char *Str) {
  if (Str != NULL) {
    while (*Str != '\0') {
      if (*Str == '+') {
        *Str = ' ';
        }
      ++Str;
      }
    }
  } /* PlusesToSpaces */

/*****************************************************************************/
/*                                                                           */
/* Function       : HexVal (STATIC)                                          */
/*                                                                           */
/* Description    : This function returns a number that corresponds          */
/*                  to the value of a character treated as                   */
/*                  a hex digit. Case insensitive. Characters outside        */
/*                  0-9,a-f,A-F have a value of 0.                           */
/*                                                                           */
/*****************************************************************************/
 static int HexVal(char c) {
  int rc;
  switch (c) {
    case '1': rc = 1;  break;
    case '2': rc = 2;  break;
    case '3': rc = 3;  break;
    case '4': rc = 4;  break;
    case '5': rc = 5;  break;
    case '6': rc = 6;  break;
    case '7': rc = 7;  break;
    case '8': rc = 8;  break;
    case '9': rc = 9;  break;
    case 'A': rc = 10; break;
    case 'a': rc = 10; break;
    case 'B': rc = 11; break;
    case 'b': rc = 11; break;
    case 'C': rc = 12; break;
    case 'c': rc = 12; break;
    case 'D': rc = 13; break;
    case 'd': rc = 13; break;
    case 'E': rc = 14; break;
    case 'e': rc = 14; break;
    case 'F': rc = 15; break;
    case 'f': rc = 15; break;
    default : rc =  0; break;
    }
  return(rc);
  }/* HexVal */
```

Figure A-9   eimdemo.c 7/10

```
/********************************************************************************/
/*                                                                              */
/* Function       : TranslateEscapes (STATIC)                                   */
/*                                                                              */
/* Description    : Translate the escape sequences induced by HTTP. The         */
/*                  sequences consist of %xx, where xx is a hex number.         */
/*                  We replace the % character with the actual character        */
/*                  (i.e., the one whose ASCII value is xx), and then           */
/*                  shift over the rest of the string to remove the xx.         */
/*                  This is done in-place.                                      */
/*                                                                              */
/********************************************************************************/
static void TranslateEscapes(char *Str) {
  char *NextEscape;
  char RealValue;
  int  AsciiValue;

  NextEscape = strchr(Str, '%');
  while (NextEscape != NULL) {
    AsciiValue = (16 * HexVal(NextEscape[1])) + HexVal(NextEscape[2]);
    *NextEscape = (char) AsciiValue;
    memmove(&NextEscape[1], &NextEscape[3],
            strlen(&NextEscape[3]) + 1);
    NextEscape = strchr(&NextEscape[1], '%');
    }
  } /* TranslateEscapes */

/********************************************************************************/
/* Function       : DisplayResults                                              */
/*                                                                              */
/* Description    : Create the web page containing the results                  */
/*                                                                        g     */
/********************************************************************************/
int DisplayResults(char *pEmployee,
                   char *pUniqueName,
                   char *pDescription,
                   char *pAliasName,
                   char *pDomainUrl,
                   char *pTgtReg,
                   char *pTgtUserName,
                   char *pBpxUserid) {

  char     *pDebug       = NULL;
```

Figure A-10   eimdemo.c 8/10

```
/* Create the first half of the web page */
  printf("Content-type: text/html\n\n");
  printf("<HTML>");
  printf("<HEAD>");
  printf("<TITLE>EIM Demo</TITLE>");
  printf("</HEAD>");
  printf("<BODY>");
  printf("<H1>EIM Demo</H1>");
  printf("<form method=POST action='/testexec/eimdemo'>");
  printf("<P>");
  printf("<PRE>");
  printf("Employee <input type='text' name='employee' size=80");
  printf(" maxlength=512 value='%s'>",pEmployee);
  printf("</PRE>");
  printf("<P>");
  printf("<PRE>");
  printf("<input type='submit' name='pushbutton' value='Access Data'>");
  printf("<input type='reset' name='pushbutton' value='Reset'>");
  printf("</PRE>");
  printf("<HR>");
  printf("<H3>Employee Information</H3>");
  printf("<P>");
  printf("<PRE>");
  printf("Unique Name <input type='text' name='uName' size=80");
  printf(" maxlength=512 value='%s'>",pUniqueName);
  printf("</PRE>");
  printf("<PRE>");
  printf("Description <input type='text' name='description' size=80");
  printf(" maxlength=512 value='%s'>",pDescription);
  printf("</PRE>");
  printf("<PRE>");
  printf("Other Name(s) <input type='text' name='aliasName' size=80");
  printf(" maxlength=512 value='%s'>",pAliasName);
  printf("</PRE>");
  printf("<P>");
  printf("<PRE>");
  printf("EIM Domain  <input type='text' name='eimDomain' size=80");
  printf(" maxlength=512 value='%s'>",pDomainUrl);
  printf("</PRE>");
  printf("<PRE>");
  printf("Target Registry <input type='text' name='tgtReg' size=80");
  printf(" maxlength=512 value='%s'>",pTgtReg);
  printf("</PRE>");
  printf("<PRE>");
  printf("tgtUserName <input type='text' name='tgtUserName' size=80");
  printf(" maxlength=512 value='%s'>",pTgtUserName);
  printf("</PRE>");
  printf("<P>");
  printf("<PRE>");
  printf("CGI User ID <input type='text' name='cgiUserid' size=80");
  printf(" maxlength=512 value='%s'>",pBpxUserid);
  printf("</PRE>");
  printf("<P>");
  printf("<PRE>");
  return(TRUE);
  }/* DisplayResults*/
```

Figure A-11 eimdemo.c 9/10

```
/*****************************************************************************/
/* Function      : DisplayResults2                                          */
/*                                                                          */
/* Description   : Create second half of the dynamic web page               */
/*                                                                     g    */
/*****************************************************************************/
int DisplayResults2() {
char     *pDebug       = NULL;
printf("<P>");
  printf("No additional information for this employee.");
  printf("</BODY>");
  printf("</HTML>");
  return(TRUE);
  } /* DisplayResults2 */

/*************************************************************/
/* This function will output a message if an error occurs when */
/* attempting to display a HTML page.                        */
/*************************************************************/
int ErrMsg (char *msg, char  *pValue)
  {
  printf("Content-type: text/html\n\n");
  printf("<html>\n");
  printf("<head>\n");
  printf("<title>Error</title>\n");
  printf("</head>\n");
  printf("<body>\n");
  printf("<h1>Error</h1>\n");
  printf("<hr>\n");
  printf("<p>\n");
  printf("An error occurred in the CGI program.\n");
  printf("The specific error message is shown below:\n");
  printf("<p>\n");
  printf("<pre>%s</pre>\n<p>\n", msg);
  printf("The value in error is shown below:\n");
  printf("<pre>%s</pre>\n", pValue);
  printf("<p>\n");
  printf("<hr>\n");
  printf("</body>\n");
  printf("</html>\n");
  return(TRUE);
  } /* ErrMsg */
/*************************************************************/
/* This function will output a message if an error occurs when */
/* attempting to call an EIM API.                            */
/*************************************************************/
int EimErr (char *msg, int rc, EimRC *err,
            char *pDomainUrl, char *pTgtReg,
            char *pBindDn, char *pBindPw) {
  printf("Content-type: text/html\n\n");
  printf("<html>\n");
  printf("<head>\n");
  printf("<title>Error</title>\n");
  printf("</head>\n");
  printf("<body>\n");
  printf("<h1>Error</h1>\n");
  printf("<hr>\n");
  printf("<p>\n");
  printf("An error occurred in the CGI program.\n");
  printf("The specific error message is shown below:\n");
  printf("<p>\n");
  printf("<pre>%s:%i</pre>\n<p>\n", msg, rc);
  printf("The error string is:\n");
  printf("   %s\n",eimErr2String(err));
  printf("<p>\n");
  printf("The domain is: %s\n",pDomainUrl);
  printf("<p>\n");
  printf("The target reg is: %s\n",pTgtReg);
  printf("<p>\n");
  printf("The bind Dn is: %s\n",pBindDn);
  printf("<p>\n");
  printf("The bind Pw is: %s\n",pBindPw);
  printf("<p>\n");
  printf("<hr>\n");
  printf("</body>\n");
  printf("</html>\n");
  return(TRUE);
  } /* EimErr */
```

Figure A-12 eimdemo.c 10/10

eimdemo2.c

```
/* Includes */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#define TRUE 1

/* Function definition                                         */
int         DisplayResults (char *, char *, char *);

extern char **environ;

/****************************************************************/
/*                                                              */
/* Function       : main                                        */
/*                                                              */
/* Description    : This is a spawned program which opens a     */
/*                  file owned by the current user id           */
/*                                                              */
/****************************************************************/
main(int argc, char *argv[]) {

  int   rc = 0;

  /* Environment variables of interest                         */
  char *pBpxUserid;                /* _BPX_USERID              */

  char              fileName[100];
  char              fileContent[80+1];
  FILE              *fp                = NULL;

  /* Build the file name: /u/...userid.../test.data           */
  bzero(&fileName,sizeof(fileName));
  pBpxUserid  = getenv("_BPX_USERID");
  strcat(fileName,"/u/");
  strcat(fileName,pBpxUserid);
  strcat(fileName,"/test.data");

  /* Read the file                                             */
  fp = fopen(fileName, "r");
  bzero(&fileContent,sizeof(fileContent));
  fgets(fileContent,80,fp);
  rc = fclose(fp);

  /* Write the bottom half of the dynamic web page             */
  rc = DisplayResults(pBpxUserid,fileName,fileContent);

  } /* main */
```

Figure A-13 eimdemo2.c 1/2

```
/*****************************************************************************/
/* Function       : DisplayResults                                           */
/*                                                                           */
/* Description    : Create second half of the dynamic web page               */
/*                                                                     g     */
/*****************************************************************************/
int DisplayResults(char  *pBpxUserid,
                   char *pFileName,
                   char *pFileContent) {

  char     *pDebug       = NULL;

  printf("<HR>");
  printf("<P>");
  printf("<PRE>");
  printf("Spawned program userid:<input type='text' name='bpxUserid'");
  printf("size=80 maxlength=1024 value='%s'>",pBpxUserid);
  printf("</PRE>");
  printf("<P>");
  printf("<PRE>");
  printf("File Name: <input type='text' name='fileName'");
  printf("size=80 maxlength=1024 value='%s'>",pFileName);
  printf("</PRE>");
  printf("<P>");
  printf("<PRE>");
  printf("File Content: <input type='text' name='fileContent'");
  printf("size=80 maxlength=1024 value='%s'>",pFileContent);
  printf("</PRE>");
  printf("</BODY>");
  printf("</HTML>");
  return(TRUE);
  } /* DisplayResults */
```

Figure A-14 eimdemo2.c 2/2

eimdemo3.c

```
/* Includes */
#undef  environ
#define _SHARE_EXT_VARS
#include <errno.h>
#include <spawn.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <eim.h>
#define _POSIX_SOURCE
#include <unistd.h>
#include <sys/wait.h>
#define TRUE 1
/* Definition of structure used in linked list of variables */
typedef struct _argument {
  char             *VariableName;
  char             *Value;
  struct _argument *pNext;
  } PARMLIST, *PPARMLIST;

/* Parms required by __spawnp2() */
char          *file;
int            fd_count;
int           (*pFd_array)[];
struct __inheritance inherit;
pid_t          pid, pid2;
int           *pStatus = NULL;

/* Function definitions */
int DisplayResults (char *, char *, char *, char *, char *);
int         ErrMsg (char *msg, char *pValue);
PPARMLIST   ReadArguments(int InputLength);
static void ASCIIToEBCDIC(char *Str);
static void PlusesToSpaces(char *Str);
static int  HexVal(char c);
static void TranslateEscapes(char *Str);
static void itoa(int, char *);
```

Figure A-15   eimdemo3.c 1/10

```
/******************************************************************************/
/*                                                                            */
/* Function       : main                                                      */
/*                                                                            */
/* Description    : This is a CGI program which takes the output of a form    */
/*                  submitted with a method of POST, and displays a list      */
/*                  of the variable names and values.                         */
/*                                                                            */
/******************************************************************************/
main(int argc, char *argv[]) {

  char     *requestMethod;         /* copies of environment vars   */
  char     *contentLength;
  char     *pRemoteHost;           /* REMOTE_HOST environment var  */
  char     *pRemoteUser;           /* REMOTE_USER environment var  */
  char     *pPassword;             /* PASSWORD    environment var  */
  char     *pBpxUserid;            /* _BPX_USERID environment var  */

  int       argLength;             /* linked list of field values  */
  PPARMLIST pParm        = NULL;
  PPARMLIST pHead        = NULL;
int            i;
  int            rc           = 0;/* EIM variables               */
  char           eimerr[200];
  EimRC         *err           = (EimRC *)eimerr;
  EimHandle      handle;
  EimConnectInfo connInfo;
  EimIdentifierInfo idName;
  char               listData[4000];
  EimList           *pListTgtUserNames = NULL;
  EimTargetIdentity *pEntry            = NULL;
  EimListData       *pListUserName     = NULL;
  char              *pUserName         = NULL;
  int                userNameLength    = 0;
  char              *pFileName         = NULL;
  char              *pEmployee         = NULL;
  char              *pAccessUser       = NULL;
  char              fileContent[80+1];
  FILE              *fp                = NULL;
  char              *pDomainURL        = NULL;
  char              *pSrcReg           = NULL;
  char              *pSrcUserName      = NULL;
  char              *pTgtReg           = NULL;
  char              *pTgtAddlInfo      = NULL;
  char               tgtUserName[9];
  char              *args[20];
  int                pd[2];
  int                fclen;

  /* Get request method - only POST is recognized by this program */
  /* -- POST must be in uppercase (this avoids use of stricmp)    */
  requestMethod = getenv("REQUEST_METHOD");
  if ((requestMethod == NULL) || (strcmp(requestMethod, "POST"))) {
    ErrMsg("REQUEST_METHOD environment variable not properly set to
POST\n",requestMethod);
    return(0);
    }
```

Figure A-16   eimdemo3.c 2/10

```
/* Get the arguments passed in to this program */
  contentLength = getenv("CONTENT_LENGTH");
  if (contentLength == NULL) {
    ErrMsg("CONTENT_LENGTH environment variable not set\n",NULL);
    return(0);
    }
  argLength = atoi(contentLength);
  pHead = ReadArguments(argLength);
  pParm = pHead;

  /* Extract field values from parameter linked list           */
  while (pParm) {
    if (!strcmp(pParm->VariableName,"employee"))
      pEmployee = pParm->Value;
    else if (!strcmp(pParm->VariableName,"fileName"))
      pFileName = pParm->Value;

    pParm = pParm->pNext;
    }

  /* Get the employee's local user id...*/
  err->memoryProvidedByCaller = 200;

  pDomainURL = "ldap://alps4001.pok.ibm.com/ibm-eimdomainname=MyDomain,c=us";
  rc = eimCreateHandle(&handle,pDomainURL,err);
  if (rc) { fprintf(stderr,"Create Handle failed: %i\n",rc); }
  connInfo.type = EIM_SIMPLE;
  connInfo.creds.simpleCreds.protect = EIM_PROTECT_CRAM_MD5;
  connInfo.creds.simpleCreds.bindDn  = "cn=administrator";
  connInfo.creds.simpleCreds.bindPw  = "secret";
  rc=eimConnect(&handle,connInfo,err);
  if (rc) { fprintf(stderr,"eimConnect failed: %i\n",rc); }
```

Figure A-17 eimdemo3.c 3/10

```
pListTgtUserNames = (EimList *)listData;
  idName.id.name    = pEmployee;
  idName.idtype     = EIM_NAME;
  pTgtReg           = "MyLocalReg";
  rc=eimGetTargetFromIdentifier(&handle,&idName,pTgtReg,NULL,
                                4000,pListTgtUserNames,err);
  if (rc) { fprintf(stderr,"eimGetTargetFromIdentifier failed:%i\n",rc);
            fprintf(stderr,"   %s\n",eimErr2String(err));
            fprintf(stderr,"   handle : %p\n",&handle);
            fprintf(stderr,"      idName.id.name: %s\n",idName.id.name);
            fprintf(stderr,"      pTgtReg: %s\n",pTgtReg);
            fprintf(stderr,
                    "      bindDn: %s\n",connInfo.creds.simpleCreds.bindDn);
            return(0);
          }

  /* convert the user names in the EimList structure to a string */
  memcpy(tgtUserName,0x00,9);
  pEntry = (EimTargetIdentity *)
           ((char *)pListTgtUserNames + pListTgtUserNames->firstEntry);
  for (i=0; i<pListTgtUserNames->entriesReturned; i++) {

    userNameLength = pEntry->userName.length;
    pUserName      = (char *)pEntry + pEntry->userName.disp;
    strcpy(tgtUserName,pUserName);

    pEntry = pEntry + pEntry->nextEntry;
    }
  fprintf(stderr,"---->tgtUserName: %s\n",tgtUserName);

  rc=eimDestroyHandle(&handle,err);
  if (rc) { fprintf(stderr,"eimDestroyHandle failed: %i\n",rc); }


  /* Create a pipe for communicating between cgi and spawned pgms */
  rc = pipe(pd);
  if (rc) { fprintf(stderr,"pipe failed: %i\n",rc); }
  fprintf(stderr,"--->pd 0 : %i\n",pd[0]);
  fprintf(stderr,"--->pd 1 : %i\n",pd[1]);
```
Figure A-18 eimdemo3.c 4/10

```
/* spawn a process under the target user id which            */
  file="/u/WEBSRV/testexec/eimdemo2";
  fd_count = pd[1] + 1;
  pFd_array = malloc(sizeof(int) * fd_count);
  for (i=0; i < fd_count-1; i++) {
    (*pFd_array)[i] = SPAWN_FDCLOSED;
    }
  (*pFd_array)[2] = 2; /* the spawned pgm will to stderr       */
  (*pFd_array)[i] = 1; /* the spawned pgm will write to pipe w/*/
                       /* stdout                               */

  bzero(&inherit, sizeof(struct __inheritance));
  inherit.flags = SPAWN_SETUSERID;
  strcpy(inherit.userid,tgtUserName);
  fprintf(stderr,"--->>inherit.userid >%s<\n",inherit.userid);
  args[0] = NULL;

  pid  = __spawnp2(file, fd_count, *pFd_array, &inherit,
                (const char **) args, (const char **) environ);
  if (pid == -1) {
    fprintf(stderr,"__spawnp2 failed: %i %s\n", errno, strerror(errno));
    }

  /* Get info from pipe                                        */
  rc = close(pd[1]);  /* close write end as it is not used here */
  pid2 = wait(pStatus); /* wait for spawned process to end. */
  fprintf(stderr,"wait \n");
  fprintf(stderr,"-->errno=%d, errno2=%8.8x - %s\n", errno,
                  __errno2(), strerror(errno));
  if (WIFEXITED(*pStatus))
    fprintf(stderr,"-->wait exited normally\n");
  fclen = read(pd[0],fileContent, sizeof(fileContent));
  if (fclen <=0) {
    fprintf(stderr,"pipe read failed: %i\n",fclen);
    fprintf(stderr,"\terrno=%d, errno2=%8.8x - %s\n", errno,
                  __errno2(), strerror(errno));
    }
  fprintf(stderr,"----->fileContent: %s\n",fileContent);
  rc = close(pd[0]);
  if (rc) { fprintf(stderr,"pipe close failed: %i\n",rc); }

  /* Get environment variables                                 */
  pBpxUserid  = getenv("_BPX_USERID");

  /* Return the results in an HTML document */
  rc = DisplayResults(pEmployee, tgtUserName,
                    pFileName, fileContent, pBpxUserid);
  } /* main */
```

Figure A-19 eimdemo3.c 5/10

```
/*****************************************************************************/
/*                                                                           */
/* Function       : ReadArguments                                            */
/*                                                                           */
/* Description    : Read the arguments from stdin that are supplied          */
/*                  to a CGI program when the method is POST.                */
/*                  Breaks up the input into                                 */
/*                  (Variable, Value) pairs.                                 */
/*                  Handles translating of all the special characters        */
/*                  that HTTP puts into the strings.                         */
/*                                                                           */
/*****************************************************************************/

PPARMLIST ReadArguments(int InputLength){
  PPARMLIST pCur  = NULL;
  PPARMLIST pHead = NULL;
  PPARMLIST pPrev = NULL;
  char     *pInput;
  char     *pToken;

  /* Read in the input                             */
  /* - ck for 1) no input parms, 2) malloc failure */
  if (InputLength < 1) {
    return(NULL);
    }
  pInput = malloc(InputLength + 1);
  if (pInput == NULL) {
    return(NULL);
    }
  gets(pInput);

  /* Variables are separated by the "&" character */
  pToken = strtok(pInput, "&");

  while (pToken) {
    /* Create and fill in linked list of variable information */
    pCur = malloc(sizeof(PARMLIST));
    pCur->VariableName = pToken;
    pToken = strchr(pToken, '=');
    if (pToken) {
      *pToken = '\0';           /* overwrite = with a NULL */
      pCur->Value = ++pToken;
      PlusesToSpaces( pToken );
      TranslateEscapes( pToken );
      ASCIIToEBCDIC( pToken );
      }
    else {
      pCur->Value = NULL;
      }
    if (pPrev) {
      pPrev->pNext = pCur;
      }

    if (!pHead) {
      pHead = pCur;
      }
    pPrev = pCur;
    pToken = strtok(NULL, "&");
    }

  if (pHead) {
    pPrev->pNext = NULL;
    }
  return(pHead);
  } /* ReadArguments */
```

Figure A-20 eimdemo3.c 6/10

```
/*****************************************************************************/
/*                                                                           */
/* Function       : ASCIIToEBCDIC  (STATIC)                                  */
/*                                                                           */
/* Description    : This one's easy. It just translates any '+'              */
/*                                                                           */
/*****************************************************************************/

static void ASCIIToEBCDIC(char *Str) {
  if (Str != NULL) {
    while (*Str != '\0') {
      switch (*Str) {
        case 0x2F : *Str = '/'; break;
        default   :             break;
        }
      ++Str;
      }
    }
  } /* ASCIIToEBCDIC */

/*****************************************************************************/
/*                                                                           */
/* Function       : PlusesToSpaces (STATIC)                                  */
/*                                                                           */
/* Description    : This one's easy. It just translates any '+'              */
/*                  characters found into ' ' characters.                    */
/*                                                                           */
/*****************************************************************************/

static void PlusesToSpaces(char *Str) {
  if (Str != NULL) {
    while (*Str != '\0') {
      if (*Str == '+') {
        *Str = ' ';
        }
      ++Str;
      }
    }
  } /* PlusesToSpaces */
```

Figure A-21   eimdemo3.c 7/10

```
/******************************************************************************/
/*                                                                            */
/* Function       : HexVal (STATIC)                                           */
/*                                                                            */
/* Description    : This function returns a number that corresponds          */
/*                  to the value of a character treated as                    */
/*                  a hex digit. Case insensitive. Characters outside         */
/*                  0-9,a-f,A-F have a value of 0.                            */
/*                                                                            */
/******************************************************************************/
 static int HexVal(char c) {
  int rc;

  switch (c) {
    case '1': rc = 1;  break;
    case '2': rc = 2;  break;
    case '3': rc = 3;  break;
    case '4': rc = 4;  break;
    case '5': rc = 5;  break;
    case '6': rc = 6;  break;
    case '7': rc = 7;  break;
    case '8': rc = 8;  break;
    case '9': rc = 9;  break;
    case 'A': rc = 10; break;
    case 'a': rc = 10; break;
    case 'B': rc = 11; break;
    case 'b': rc = 11; break;
    case 'C': rc = 12; break;
    case 'c': rc = 12; break;
    case 'D': rc = 13; break;
    case 'd': rc = 13; break;
    case 'E': rc = 14; break;
    case 'e': rc = 14; break;
    case 'F': rc = 15; break;
    case 'f': rc = 15; break;
    default : rc =  0; break;
    }

  return(rc);
  } /* HexVal */

/******************************************************************************/
/*                                                                            */
/* Function       : TranslateEscapes (STATIC)                                 */
/*                                                                            */
/* Description    : Translate the escape sequences induced by HTTP. The       */
/*                  sequences consist of %xx, where xx is a hex number.       */
/*                  We replace the % character with the actual character      */
/*                  (i.e., the one whose ASCII value is xx), and then         */
/*                  shift over the rest of the string to remove the xx.       */
/*                  This is done in-place.                                    */
/*                                                                            */
/******************************************************************************/
static void TranslateEscapes(char *Str) {
  char *NextEscape;
  char RealValue;
  int  AsciiValue;

  NextEscape = strchr(Str, '%');
  while (NextEscape != NULL) {
    AsciiValue = (16 * HexVal(NextEscape[1])) + HexVal(NextEscape[2]);
    *NextEscape = (char) AsciiValue;
    memmove(&NextEscape[1], &NextEscape[3],
            strlen(&NextEscape[3]) + 1);
    NextEscape = strchr(&NextEscape[1], '%');
    }
  } /* TranslateEscapes */
```

Figure A-22 eimdemo3c.8/10

```
/****************************************************************************/
/* Function       : DisplayResults                                         */
/*                                                                         */
/* Description    : Create the web page containing the results              */
/*                                                                    g     */
/****************************************************************************/
int DisplayResults (char *pEmployee,
                    char *ptgtUserName,
                    char *pFileName,
                    char *pFileContent,
                    char *pBpxUserid) {

  char     *pDebug       = NULL;

  /* Create and Write the web page. */
  printf("Content-type: text/html\n\n");
  printf("<HTML>");
  printf("<HEAD>");
  printf("<TITLE>EIM Lookup Demo</TITLE>");
  printf("</HEAD>");
  printf("<BODY>");
  printf("<H1>EIM Lookup Demo</H1>");
  printf("<form method=POST action='/testexec/eimdemo'>");
  printf("<P>");
  printf("<PRE>");
  printf("Employee<input type='text' name='fileName' size=40");
  printf(" maxlength=512 value='%s'>",pEmployee);
  printf("<P>");
  printf("<PRE>");
  printf("tgtUserName <input type='text' name='tgtUserName' size=40");
  printf(" maxlength=512 value='%s'>",ptgtUserName);
  printf("<P>");
  printf("<PRE>");
  printf("Enter the file name <input type='text' name='fileName' size=40");
  printf(" maxlength=512 value='%s'>",pFileName);
  printf("</PRE>");
  printf("<P>");
  printf("<PRE>");
  printf("File contains <input type='text' name='fileContent' size=40 ");
  printf("maxlength=512 value='%s'>",pFileContent);
  printf("</PRE>");
  printf("<P>");
  printf("<PRE>");
  printf("_BPX_USERID:<input type='text' name='bpxUserid'");
  printf("size=40 maxlength=1024 value='%s'>",pBpxUserid);
  printf("</PRE>");
  printf("<P>");
  printf("<PRE>");
  printf("<input type='submit' name='pushbutton' value='Lookup'>");
  printf("<input type='reset' name='pushbutton' value='Reset'>");
  printf("<HR>");
  printf("</PRE>");
  printf("</BODY>");
  printf("</HTML>");
  return(TRUE);
  } /* DisplayResults */
```

Figure A-23 eimdemo3.c 9/10

```
/*****************************************************************************/
/* This function will output a message if an error occurs when attempting     */
/* to display a HTML page.                                                    */
/*****************************************************************************/
int ErrMsg (char *msg, char  *pValue)
  {
  printf("Content-type: text/html\n\n");
  printf("<html>\n");
  printf("<head>\n");
  printf("<title>Error</title>\n");
  printf("</head>\n");
  printf("<body>\n");
  printf("<h1>Error</h1>\n");
  printf("<hr>\n");
  printf("<p>\n");
  printf("An error occurred in the CGI program.\n");
  printf("The specific error message is shown below:\n");
  printf("<p>\n");
  printf("<pre>%s</pre>\n<p>\n", msg);
  printf("The value in error is shown below:\n");
  printf("<pre>%s</pre>\n", pValue);
  printf("<p>\n");
  printf("<hr>\n");
  printf("</body>\n");
  printf("</html>\n");
  return(TRUE);
  } /* ErrMsg */
/*****************************************************************************/
/* Function       : itoa                                                      */
/*                                                                            */
/* Description    :                                                           */
/*                                                                  g         */
/*****************************************************************************/
void itoa (int n, char s[80]) {
  int i, c, j;
  int sign = 0;


  bzero(s, sizeof(s));
  /* save the sign of the number */
  if (n < 0) {
    sign = n;
    n = -n;
    }

  /* Convert the number to char, chars are reversed */
  i=0;
  do {
    s[i++] = n % 10 + '0';
  } while ((n /= 10) > 0);

  /* add the sign to the string */
  if (sign<0)
    s[i++] = '-';
  s[i++] = '\0';

  /* put the chars in the right order */
  for (i=0,j=strlen(s)-1; i<j; i++, j--) {
    c    = s[i];
    s[i] = s[j];
    s[j] = c;
  }
} /* itoa */
```

Figure A-24 eimdemo3.c 10/10

B

Sample test programs for PADS enhancements and RACF Password Enveloping

PADS enhancement sample code

The assembler source programs provided in this section have been used to test the PADS enhancements in 2.4.1, “PADS enhancement” on page 20.

```
ATTAFLAT CSECT
         STM   14,12,12(13)
         LR    12,15
         USING ATTAFLAT,12
         WTO   'YURY -- IN LOADED MODULE'
         LM    14,12,12(13)
         SR    15,15
         BR    14
         END   ATTAFLAT
```

Figure B-1 ATTAFLAT

```
ATTAMAIN CSECT
         SAVE  (14,12)
         LR    12,15
         USING ATTAMAIN,12
         ST    13,SAVE+4
         LA    13,SAVE
*
         WTO   'YURY -- WE ARE IN ATTAMAIN'
         LINK  EP=ATTATASK
*
         L     13,SAVE+4
         LM    14,12,12(13)
         SR    15,15
         BR    14
SAVE     DS    18F
*
         END   ATTAMAIN
```

Figure B-2 ATTAMAIN

```
ATTASUB1 CSECT
         STM   14,12,12(13)
         LR    12,15
         USING ATTASUB1,12
         ST    13,SAVE+4
         LA    13,SAVE
         LR    2,1
         WTO   'YURY -- WE ARE IN SUBTASK'
         LR    15,2
         BALR  14,15
*        DC    H'0'       **********************ABEND OC1********
         L     13,SAVE+4
         LM    14,12,12(13)
         SR    15,15
         BR    14
SAVE     DS    18F
         END   ATTASUB1
```

Figure B-3 ATTASUB1

```
ATTATASK CSECT
         PRINT OFF
         COPY  EQUES
         PRINT ON,NOGEN
         STM   R14,R12,12(R13)
         LR    R12,R15
         USING ATTATASK,R12
         ST    R13,SAVE+4
         LA    R13,SAVE
* LET US TRY TO OPEN A DATA SET WHICH IS 'PADS' PROTECTED
         OPEN  (IDCB,(INPUT))
* LET US DO SOME PROGRAM CONTROL STUFF HERE
         LOAD  EPLOC=EXTMODNM
         ST    R0,LOADMODA
*
         L     R1,LOADMODA
         ATTACH EP=ATTASUB1,ECB=MAINECB
         LTR   R11,R15
         BZ    SAVETCB
         WTO   'YURY - ATTACH IS NOT SUCCESSFUL',ROUTCDE=11
         B     EXIT
SAVETCB  ST    R1,TCBADDR
         L     R1,124(R1)       A(JSTCB)
         ST    1,JSTCBADR
*
         WAIT   ECB=MAINECB
         DETACH TCBADDR

------- continued in next figure ----------
```

Figure B-4 ATTATASK 1/2

--------- continued from last figure --------

```
*                                                                       00001198
* NOW WE CAN READ A LITTLE FROM PADS PROTECTED DATA SET                 00001199
*                                                                       00001200
READLOOP GET   IDCB,AREA                                                00001210
         AP    RECNUM(3),ONE(1)                                         00001300
         B     READLOOP                                                 00001400
EOFINPUT UNPK  OUTNUM(5),RECNUM(3)                                      00001500
         OI    OUTNUM+4,X'F0'                                           00001600
         WTO   MF=(E,SAYRECNM)                                          00001700
         SR    R11,R11                                                  00001710
EXIT     EQU   *                                                        00003600
         CLOSE (IDCB)                                                   00003610
         DELETE EPLOC=EXTMODNM                                          00003700
         L     R13,SAVE+4                                               00003800
         LR    R15,R11                                                  00003900
         RETURN (14,12),RC=(15)                                         00004000
         DROP  R12                                                      00004100
*                                                                       00004200
SAVE     DS    18F                                                      00004300
EXTMODNM DC    CL8'ATTAFLAT'                                            00004400
LOADMODA DS    A                                                        00004500
TCBADDR  DS    A                                                        00004600
JSTCBADR DS    A                                                        00004700
MAINECB  DC    F'0'                                                     00004800
IDCB     DCB   DDNAME=INPUTDS,DSORG=PS,MACRF=GM,EODAD=EOFINPUT,        *00004900
               LRECL=80                                                 00005000
RECNUM   DC    PL3'0'                                                   00005100
ONE      DC    P'1'                                                     00005200
SAYRECNM DS    0F                                                       00005300
         DC    AL2(26)                                                  00005400
         DC    AL2(0)                                                   00005500
         DC    C'YURY -- RECNUM = '                                     00005600
OUTNUM   DC    CL5' '                                                   00005700
AREA     DS    XL256                                                    00005800
         END   ATTATASK
```

Figure B-5 ATTATASK 2/2

```
R0       EQU   0                                                    00000100
R1       EQU   1                                                    00000200
R2       EQU   2                                                    00000300
R3       EQU   3                                                    00000400
R4       EQU   4                                                    00000500
R5       EQU   5                                                    00000600
R6       EQU   6                                                    00000700
R7       EQU   7                                                    00000800
R8       EQU   8                                                    00000900
R9       EQU   9                                                    00001000
R10      EQU   10                                                   00001100
R11      EQU   11                                                   00001200
R12      EQU   12                                                   00001300
R13      EQU   13                                                   00001400
R14      EQU   14                                                   00001500
R15      EQU   15                                                   00001600
```

Figure B-6 Equates

RACF Password Envelopping sample test

This is the assembler source program that has been used in 6.3.2, "Retrieving the enveloped password" on page 157, to extract the password envelope from the RACF database.

```
PWDENVLP CSECT
PWDENVLP AMODE 31
PWDENVLP RMODE 24
         PRINT NOGEN
*
SUBPNUM  EQU   1             SUBPOOL FOR ENVELOPE
*
         STM   14,12,12(13)
         LR    12,15
         USING PWDENVLP,12
         ST    13,SAVE+4
         LA    13,SAVE
*
         L     2,0(1)      A(PARM)
         ICM   3,1,1(2)
         BZ    MOVYURY     THERE IS NO A USERID
         CLI   1(2),8      IS IT > 8 CHARS ?
         BNH   MOVID       ...NO - COPY THE VALUE TO PARAMS
MOVYURY  MVC   FUNCUSR(8),=CL8'YURY'
         B     OPENSNAP
MOVID    BCTR  3,0         GET MVC MODIFIER
MOVUSR   MVC   FUNCUSR(1),2(2)
         EX    3,MOVUSR
*
OPENSNAP OPEN  (SNAP,(OUTPUT),ENVELOPE,(OUTPUT))
AFTER    BCR   0,0
*
SUPUSER  MODESET MODE=SUP
*
         CALL  IRRSEQ00,(RACFWA,                                       *
               AL#SAFRC,SAFRC,                                         *
               AL#RCFRC,RACFRC,                                        *
               AL#RCFRS,RACFRSN,                                       *
               FUNCCODE,                                               *
               FUNCPARM,          NAME OF FUNCTION PARAMETER LIST      *
               USERID,            TAKE THE DEFAULT (LNGTH=0)           *
               ACEEPTR,                                                *
               SUBPOOLN,          USE SP=1                             *
               OUTPAREA),VL
         L     11,SAFRC           SAVE RETURN CODE
         MODESET MODE=PROB
         LTR   11,11              WAS THE CALL SUCCESSFUL?
         BZ    EXTR               ...YES, LET'S PUT THE ENVELOPE
         WTO   'PWDENVLP -- UNSUCCESSFUL ENVELOPE EXTRACTION. SEE SNAP'
         SNAP  DCB=SNAP,STORAGE=(SAFRC,RACFRSN+3),ID=99
         B     EXIT
------ continued in next figure ----------------
```

Figure B-7   PWDENVLP 1/2

```
--------- continued from previous figure -------------
*
* LET US PUT THE ENCRYPTED ENVELOPE INTO A SNAP DATA SET
*
EXTR     EQU   *
         L     4,OUTPAREA         ADDRESS OF THE ENVELOPE
         L     3,4(4)             KEEP THE CONTENT OF THE ORIGINAL
         LA    2,4(3)             BUILD RDW IN REG2
         XC    4(4,4),4(4)        BUILD RDW
         STH   2,4(4)               IN THE RETURNED AREA
         NI    5(4),X'C0'         CUT LAST 96 BYTES
AFTER1   N     4,MASK32
         PRINT GEN
         PUT   ENVELOPE,4(4)
         PRINT NOGEN
         ST    3,4(4)             RESTORE THE AREA
         LH    2,6(4)
         STORAGE RELEASE,LENGTH=(2),ADDR=(4),SP=SUBPNUM
         SR    11,11
EXIT     CLOSE (SNAP,,ENVELOPE)
         L     13,SAVE+4
         LR    15,11
         RETURN (14,12),RC=(15)
*
SAVE     DS    18F
MASK32   DC    X'7FFFFFFF'
AL#SAFRC DC    F'0'
AL#RCFRC DC    F'0'
AL#RCFRS DC    F'0'
SAFRC    DC    F'0'
RACFRC   DC    F'0'
RACFRSN  DC    F'0'
ACEEPTR  DC    F'0'
OUTPAREA DC    A(0)
FUNCCODE DC    AL1(ADMN_XTR_PWENV)
SUBPOOLN DC    AL1(SUBPNUM)
USERID   DC    AL1(0),CL8' '
         DS    0F
FUNCPARM DC    AL1(4)
FUNCUSR  DC    CL8' '
         DC    X'00'
         DC    H'0',H'0'   RESERVED FOR ADMN_XTR_PWENV
         DS    4F          CREATE SOME CUSHION
ENVELOPE DCB   DDNAME=ENVELOPE,DSORG=PS,MACRF=PM,RECFM=VB,BLKSIZE=6144,*
               LRECL=6140
SNAP     DCB   DDNAME=SNAP,DSORG=PS,MACRF=W,RECFM=VBA,BLKSIZE=882,     *
               LRECL=125
RACFWA   DS    4XL256
*
         IRRPCOMP DSECT=YES  BUILDS THE 'COMP' DSECT
*
         END   PWDENVLP
```

Figure B-8 PWDENVLP 2/2

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 about ordering these publications, see "How to get IBM Redbooks" on page 299. Note that some of the documents referenced here may be available in softcopy only.

- *Exploiting S/390 Hardware Cryptography with Trusted Key Entry*, SG24-5455
- *IBM @server zSeries 990 (z990) Cryptography Implementation*, SG24-7070
- *Implementing VPNs in a z/OS Environment*, SG24-6530
- *MVS/ESA OpenEdition DCE: RACF and DCE Security Interoperation*, GG24-2526
- *OS/390 Security Server 1999 Updates Installation and Implementation Guide*, SG24-5629
- *OS/390 Security Server 1999 Updates Technical Presentation Guide*, SG24-5627*S/390 Crypto PCI Implementation Guide*, SG24-5942
- *RACF Support for Open Systems Technical Presentation*, GG26-2005
- *Stay Cool on OS/390: Installing Firewall Technology*, SG24-2046
- *zSeries Crypto Guide Update*, SG24-687

These publications are also relevant as further information sources:

- *z/OS ICSF Overview*, SA22-7519
- *z/OS ICSF System Programmer's Guide*, SA22-7520
- *z/OS ICSF Application Programmer's Guide*, SA22-7522
- *z/OS ICSF Administrator's Guide*, SA22-7521
- *z/OS ICSF Messages*, SA22-7523
- *z/OS Trusted Key Entry Workstation User's Guide 2000*, SA22-7524
- *z/OS Open Cryptographic Services Facility Application Programming*, SC24-5899
- *z/OS Open Cryptographic Services Facility Service Provider Module Developer's Guide and Reference*, SC24-5900
- *z/OS Cryptographic Services System Secure Sockets Layer Programming*, SC24-5901
- *z/OS Cryptographic Services PKI Services Guide and Reference*, SA22-7693
- *z/OS Security Server RACF Security Administrator's Guide*, SA22-7683
- *z/OS security Server RACF Command Language Reference*, SA22-7687
- *z/OS Security Server RACF Auditor's Guide*, SA22-7684
- *z/OS Security Server RACF Callable Services*, SA22-7691
- *z/OS Security Server RACF Macros and Interfaces*, SA22-7682
- *z/OS Security Server RACF System Programmers Guide*, SA22-7681
- *z/OS Security Server Firewall Technologies*, SC24-5922

- *z/OS Integrated Security Services LDAP Server Administration and Use*, SC24-5923
- *z/OS Integrated Security Services LDAP Client Programming*, SC24-5924.
- *z/OS SecureWay Security Server Open Cryptographic Enhanced Plug-ins Application Programming*, SC24-5925.
- *z/OS Integrated Security Services Enterprise Identity Mapping (EIM) Guide and Reference*, SA22-7875.
- *Policy Director Authorization Services for z/OS and OS/390 Customization and Use Version 1 Release 2*, SC24-6040.
- *z/OS IBM Ported Tools for z/OS User's Guide*, SA22-7985.
- *z/OS Communications Server IP Configuration Guide*, SC31-8775
- *z/OS Communications Server IP Configuration Reference*, SC31-8776

Online resources

These Web sites and URLs are also relevant as further information sources:

- Miscellaneous as-is tools

  `http://www-1.com/servers/eserver/zseries/zos/racf/goodies.html`
- IETF Web site

  `http://www.ietf.org`
- PKCS standards

  `http://www.rsasecurity.com/rsalabs/pkcs`
- Identrus Web site

  `http://www.identrus.com`
- z/OS certification

  `http://www.ibm.com/servers/eserver/zseries/security/ccs_certification.htm`
- IBM products certifications

  `http://www.ibm.com/security/standards/st_evaluations.shtml`
- NIST Web site

  `http://niap.nist.gov/cc-scheme`
- SUN Web site

  `http://www.sun.com`
- RACF Web site

  `http://www-1.ibm.com/servers/eserver/zseries/zos/rac`
- EIM Web site

  `http://www-1.ibm.com/servers/eserver/security/eim`
- z/OS UNIX tools and toys

  `http://www-1.ibm.com/servers/eserver/zseries/zos/unix/bpxa1toy.html`
- IETF SSH charter

  `http://www.ietf.org/html.charters/secsh-charter.html`
- OpenSSH Web site

  `http://www.openssh.com`

- PuTTY Web site

  `http://www.chiark.greenend.org.uk/~sgtatham/putty`

How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

  ibm.com/redbooks

Help from IBM

IBM Support and downloads

  ibm.com/support

IBM Global Services

  ibm.com/services

Index

IBM

Redbooks

z/OS 1.6 Security Services Update

z/OS 1.6 Security Services Update

The latest on z/OS LDAP, RACF, System SSL, OpenSSH, and more

z/OS Enterprise Identity Mapping (EIM) demonstrated

RACF Multilevel Security introduced and explained

This IBM Redbook describes the z/OS Security Services provided by z/OS as of z/OS 1.6.

As this is a vast subject and some of these services have already been addressed by other redbooks, usually in the OS/390–z/OS 1.2 time frame, we concentrate on new services or services that have gone under a noticeable evolution since they were described in previous redbooks, and we provide simple examples of utilization.

The first chapter is a summary of all Security Services available in z/OS 1.6 today.

This book addresses the enhancements in RACF Security Services in the domains of the Unix Security Services, the digital certificate handlings, and the more traditional Program Access to Data Sets (PADS) function, which have all been enhanced to better match the eBusiness needs in terms of compliance with standards and improved security. Note that RACF Multilevel Security (MLS) is also introduced in this book, with a practical example of its application to TCP/IP Security.

SG24-6448-00 ISBN 0738493538