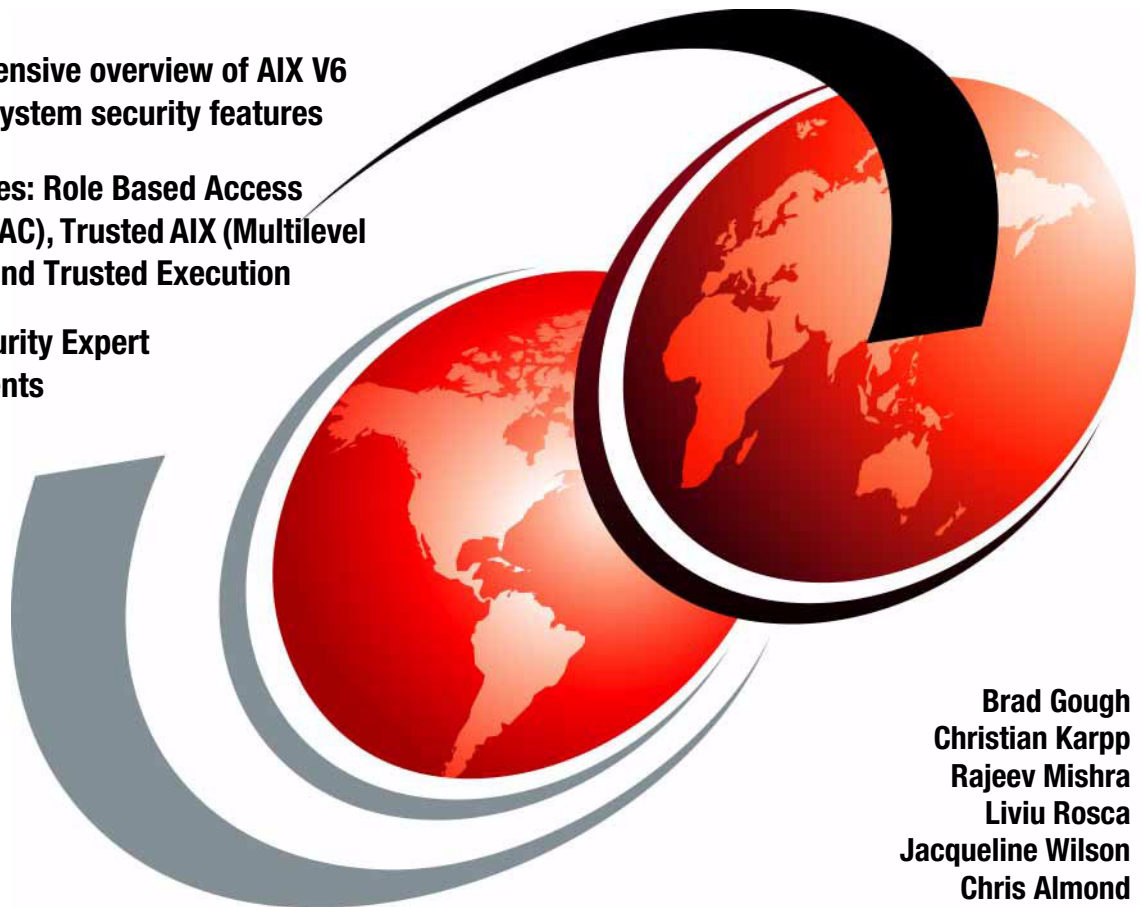**IBM**

AIX V6 Advanced Security Features
Introduction and Configuration

A comprehensive overview of AIX V6 operating system security features

New features: Role Based Access Control (RBAC), Trusted AIX (Multilevel Security), and Trusted Execution

AIX V6 Security Expert enhancements

Brad Gough
Christian Karpp
Rajeev Mishra
Liviu Rosca
Jacqueline Wilson
Chris Almond

Redbooks

ibm.com/redbooks

IBM

International Technical Support Organization

**AIX V6 Advanced Security Features
Introduction and Configuration**

September 2007

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

First Edition (September 2007)

This edition applies to IBM AIX Version 6.1.

Note: This book is based on a pre-GA version of a product and may not apply when the product becomes generally available. We recommend that you consult the product documentation or follow-on versions of this IBM Redbooks publication for more current information.

Contents

Notices

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

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

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.

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

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

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

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

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

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

COPYRIGHT LICENSE:

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

xi

Trademarks

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

AIX 5L™	MVS™	RS/6000®
AIX®	OS/390®	System p™
Argus®	OS/400®	System p5™
DB2®	POWER4™	Tivoli®
eServer™	pSeries®	z/OS®
HACMP™	Redbooks®	
IBM®	Redbooks (logo) ®	

The following terms are trademarks of other companies:

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

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

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

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

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

Preface

AIX Version 6.1 provides many significant new security technologies and security enhancements. The purpose of this IBM® Redbooks® publication is to highlight and explain the security features at the conceptual level, as well as provide practical examples of how they may be implemented. Some features are extensions of features made available in prior AIX® releases, and some are new features introduced with AIX V6.

IT managers, system architects, and system administrators will find this book useful for learning more about the different AIX V6 operating system security features, at both the conceptual and more detailed practitioner levels.

In this IBM Redbooks publication, you will find a detailed introduction to the full AIX V6 security tool set, with emphasis on the most significant extensions and additional features provided in AIX Version 6.1, which include:

► Enhancements to AIX Security Expert, a security hardening tool.

► Secure By Default (SBD).

► File Permission Manager (FPM)

► A fully implemented enhanced Role Based Access Control system (RBAC)

► Encrypted File Systems (EFS)

► Trusted AIX (an implementation of Multilevel Security or MLS)

► Trusted Execution (TE)

► Workload Partitions (WPAR)[1]

This IBM Redbooks publication provides a technical introduction to these features. The topics are both broad and very complex. This IBM Redbooks publication will serve as an initial reference for describing all of these features together in a single volume, to the security/system hardening oriented audience.

---

[1] Workload Partitions (WPARs) are a new operating system virtualization feature introduced in AIX V6. We present some security concerns related to working with WPARs. For a complete technical introduction to WPAR features in AIX V6, you should refer to the IBM Redbooks publication *Introduction to Workload Partition Management in AIX V6,* SG24-7431.

The team that wrote this book

This IBM Redbooks publication was produced by a team of specialists that came from around the world to work together at the IBM International Technical Support Organization in Austin, Texas, in April and May of 2007.

Brad Gough is an AIX technical specialist working for IBM Global Services in Sydney, Australia. Brad has been with IBM since 1997, starting as an RS/6000® hardware engineer; in 2000, he moved to AIX support. His areas of expertise include AIX, NIM, and HACMP™ on the IBM SP/2 and System p™ and System p Virtualization. He is currently involved in the installation and design of System p implementations in Australia and New Zealand. He is an IBM Certified Systems Expert – IBM System p5™ Virtualization Technical Support and IBM eServer™ p5 and pSeries® Enterprise Technical Support AIX 5L™ V5.3. This is his second IBM Redbooks publication.

Christian Karpp (CISSP) is an IT Specialist and Security Consultant at IBM in Mannheim, Germany. He has worked at IBM for 12 years, consulting on and implementing security projects with IBM customers in varied industries. Christian has an extensive background in Information Security disciplines and System p and has worked with AIX for over ten years. Christian studied Computer Science and graduated from the University of Applied Science in Mannheim, Germany.

Rajeev Mishra is a System Software Engineer at the IBM India Software Labs, Bangalore. He is currently working on the Functional Verification Testing of AIX Security. He graduated from Institute of Technology, BHU, Varanasi with a Bachelor of Technology degree in Computer Science and Engineering. He has two years of experience in IBM. He has expertise in journaled file system (JFS), enhanced journaled file system (JFS2), and various security topics in AIX.

Liviu Rosca is an IT Specialist with IBM Global Technology Services, Romania. He has been working for IBM for five years providing support for pSeries, AIX, HACMP, and WVR. His area of expertise include pSeries, AIX, HACMP, networking, security, and telecommunications. He is IBM Certified AIX 5L and HACMP System Administrator and CCNP. He teaches AIX and HACMP classes.

Jackie Wilson has recently become a member of the Advanced Technical Sales team, based in Dallas Texas as an Executive IT Specialist. She has been in AIX development since 1985, starting with writing communications firmware and device driver code. After being in positions in customer service and test architecture, she returned to development on the X.25 project. Her security focused activities started with a role as technical lead for the AIX IP Security team. Jackie has worked with many product and research teams in IBM. Jackie became a Senior Technical Staff member in 2004, assuming the position of security architect for AIX development. She has led the certification efforts for AIX 5L V5.2 and V5.3, and the AIX software encryption projects. She worked in a strategy role on the AIX security roadmap, developing AIX security plans and conceiving of projects such as AIX Security Expert. She enjoys her new job working with customers and educating field personnel on AIX security offerings.

Production of this IBM Redbooks publication was managed by:

Chris Almond, an ITSO Project Leader and IT Architect based at the ITSO Center in Austin, Texas. In his current role, he specializes in managing content development projects focused on Linux®, AIX 5L systems engineering, and other innovation programs. He has a total of 16 years of IT industry experience, including the last eight with IBM.

Acknowledgements

Also, assistance with development of CliC content was provided by **Tamas Visegrady**, **JianHua Feng**, **Mihai Togan**, **Leo Moesgaard**, **Lars Elmegaard-Fessel**, and **Dan Kyhl**.

The team would also like to acknowledge the support efforts of **Jay Kruemcke**, the System p AIX Offering Manager, **Jay Beck**, Development Manager, AIX Security Development, **Scott Vetter**, IBM Redbooks System p Team Leader, and our editor for this book, **Wade Wallace**, from the ITSO Authoring Services team.

Become a published author

Join us for a two- to six-week residency program! Help write a book dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You will have the opportunity to team with IBM technical professionals, Business Partners, and Clients.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you will 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 books to be as helpful as possible. Send us your comments about this book or other IBM Redbooks in one of the following ways:

► Use the online **Contact us** review Redbooks form found at:

ibm.com/redbooks

► Send your comments in an e-mail to:

redbooks@us.ibm.com

► Mail your comments to:

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

Part 1

AIX V6 operating system security features

Chapter 1 begins this book by providing a review of operating system security concepts and introductions to each of the significant security system features provided in AIX V6. Subsequent chapters provide more detailed, practitioner oriented discussion and examples. The book consists of the following chapters:

1

Introduction

A main goal of this chapter is to explain security concepts at a level that IT managers responsible for systems security can understand. Later chapters will focus on implementation specifics needed by security and system administrators. Therefore, the highlights of those features are presented in this chapter along with the context in which they are used. The Introduction chapter consists of the following sections:

1.1 Introduction: security in the enterprise

This chapter will describe how the new security features of AIX V6 address some issues in traditional UNIX® security models and problems encountered by businesses today. The area of computer security is a dynamic and changing world, in which enterprises must be diligent to integrate and combine many mechanisms to address varying threats. In order to create a good security policy, an organization must understand its strengths and weakness, the value of its assets, and its source of threats.

Towards the end of this chapter, we will also be reviewing important complimentary security features released in prior releases, to give a complete picture of the security relevant capabilities available to customers in all releases of AIX. The customer documentation available in the AIX information center should be consulted to review specific syntax and information for each security feature.

The security of an enterprise consists of various aspects. In regards to IT security, operating system security provides for the foundation around which the rest of the software stack can build its security. AIX historically has provided for strong operating system security. AIX V6 builds on that security and provides for multiple security features addressing varied types of threats for different operating environments.

Security threats are often thought of as activities originating from external organizations, and traditionally firewalls and DMZs have been used to set up protection for the corporate environment. Computer forensic data has consistently shown that intrusions occur most often from within an enterprise. This fact is often unsettling and some level of denial often results in computer security practices that may be inadequate. Security exposures include inadvertent misuse or misconfiguration of systems to intentional harm or theft of intellectual property. Disgruntled employees or even "moles" can wreak havoc on systems because they have a lot of knowledge on what security systems are in place. Cleartext passwords, for example, are still a common practice within the corporate network. Having a global and virtual workplace can often compound the problem, where precious intellectual property leaves the country's borders. No one program or technology can solve the problem, but it is a combination of technologies, business practices, and social engineering that must be employed together to form a complete security model.

Security solutions can often be difficult to understand or implement. One must first understand the type of threat that exists for their particular situation, and the cost of a security breach. Some describe this as the "toxicity" of the data. If the cost of losing data is high, such as violating legal requirements for protecting confidentiality and privacy, businesses must be vigilant in securing systems. AIX has created several tools to simplify and enhance the administrator's ability to secure data within the operating system.

1.2 Risk analysis

Risk analysis is a critical component of establishing any security policy. Risk analysis involves identifying assets, identifying threats, and calculating risks.

Assets can be tangible or intangible. Tangible assets include computer systems and components, physical assets, data records, and so on. Intangible assets include reputation, customer confidence, reliability, employee loyalty, privacy of information, and configuration information.

When performing risk analysis, tangible and intangible assets should be listed so a thorough risk analysis can be performed.

For each type of risk, there are countermeasures that can help mitigate risk, Risk can be measured as:

Threat x vulnerability x asset value = total risk

Total risk x countermeasure = residual risk

There are several ways a customer can choose to deal with risk.

The risk can be transferred, for example, by purchasing insurance for certain situations. The risk can be accepted, for example, by doing nothing. Or, a better alternative, the risk can be reduced by implementing an effective countermeasure. With AIX V6, we have several new features that enable customers to reduce risk by implementing countermeasures.

The types of mechanisms that are included in AIX V6.1 can be considered a group of countermeasures aimed at reducing risk by improving trust. There is some cost in implementing countermeasures, whether by increasing training, denying access at times to legitimate users who may forget or lose credentials, or for example, requiring additional computations, such as with encryption technology. The value of the assets and the cost of the damage caused by an exploited vulnerability justify the increased cost of implementing security measures.

In this overview chapter, the goal is to not only explain the technology, but also the associated cost and benefit of the technology so that professionals can evaluate the types of threats they face, the assets that are being protected, and the countermeasures available to make the best selection of security technologies offered in AIX. There have been several security enhancements recently released for AIX 5L Version 5 that provide other important security benefits, and they will be mentioned as well as the new features in AIX V6.

1.3 Types of security threats

Threats to computer systems can be classified in different categories. One major category is the *threat to system integrity*. With this type of threat, modules on a system are replaced with rogue modules that leave the system open to future access into and out of the system. It can also leave systems vulnerable to destruction of key files and data. System integrity is the basis of all other security aspects of a system. If the system integrity is compromised, all decisions and operations of that system may be compromised as well, as key system files and behavior cannot be trusted or predicted.

Another common threat to computer systems is the *threat of unauthorized access*. This can be accomplished through several mechanisms. Direct brute force attacks such as password cracking software, or more subtle attacks, such as access gained by exploiting vulnerabilities in networking software, can result in unauthorized access.

A further manifestation of the threat of unauthorized access is the *threat of unauthorized system administrator access*. Buffer overflows are a common mechanism used to gain unauthorized access. This type of attack gives a program more input data than it expects. The extra input data is crafted as an executable instruction set of some kind. When the program runs, the data area overflows into the stack area, and when the program returns, the overflowed data is executed. This is usually performed on privileged commands or executables. It causes code to be executed on the stack in the privileged state. These are commonly exploited mechanisms used to gain access to protected files or directories. Fortunately, enhancements in AIX 5L V5.3 and AIX V6.1 have features to specifically reduce or eliminate the ability to use these mechanisms through proper policy configuration, definition of authorizations, and cryptographic verification of executed code. This protection can also be achieved by isolation of resources.

The threat of *violation of user security policy* is another common threat encountered in computer systems. It is expected that users of a system will have the authority and ability to own and create files, and to restrict access to those files by other users. There is a threat that the protection mechanisms under the control of the user could be bypassed, allowing a user's directories and files to be viewed or modified without the owner's consent or knowledge. Examples of these attacks include superuser attacks and virus / Trojan horse attacks. Trojan horse attacks would include those forms of attack where an agent process acting on behalf of the user (such as a Java™ applet) violates the security policy of the user.

The *threat of violation of site security policy* is a major concern to security officers. It is expected that a site will have its own security policy that is enforced without reliance on proper user behavior or voluntary adherence to site policy.

To illustrate, it is common to have several groups of people who are using the same computer system (for example, a personnel division and an accounting group), but files associated with each group must be accessible only to those in the group. There is a threat that a user may accidentally or intentionally share a file, or a copy of a file, and allow access to a user who is not authorized to have access to the information, even though both users are authorized users of the system. Standard UNIX systems have no mechanism to prohibit this type of activity by users.

A threat that is getting more and more attention by computer professionals and business practice professionals is the *threat of lack of user accountability and auditing*. There is a threat that in the event of a breach or in normal business monitoring, a system administration would be unable to associate activity on the system with the individual responsible for initiating or permitting the activity. This threat is caused by an inadequate audit system, a failure of the audit system to protect audit files, the failure of the audit system to be enabled, or a failure of the system to properly associate a user with system activity.

1.4 AIX V6.1 security features and the threats they address

AIX Security Enhancements address the various types of threats with countermeasures built into AIX at no extra charge to the customer. These features use different security and cryptographic mechanisms to determine and maintain the desired state of a system. AIX V6.1 provides mechanisms beyond traditional UNIX mechanisms to ensure trust of the components and data within the computer system. Table 1-1 shows the threats and the features that provide countermeasures to address them.

Table 1-1   Threats and countermeasures

Threat	Feature	Notes
System Integrity	Trusted Execution	Ensures binaries are not altered and no execution of malicious code occurs.
	AIX Security Expert	Reduces exposures by setting security policy.
	Trusted AIX	Trusted computing base fundamental to Multilevel System
Unauthorized Access	AIX Security Expert	Sets policies for network, passwords.
	Role Based Access Control	Fine-grained control on privileged operations.
	Secure by Default	Reduces network access.
	File Permission Manager	Reduces setuid bits.
	Encrypted File System	Files not readable by unauthorized users.
Unauthorized System Administrator Access	Role Based Access Control	Divides root into separate roles and sets privileges for processes.
	Trusted AIX	Removes the concept of root.

Threat	Feature	Notes
Violation of Site Security Policy	AIX Security Expert	Centralized policy.
	Trusted AIX	Mandatory access controls with no concept of root.
User Accounting	AIX Security Expert	Enables Auditing if not already enabled.
	Trusted AIX	Auditing required.

1.5 Types of security

There are many components to security for both users and administrators. These components[1] are broadly described as:

Data Integrity
Data integrity ensures the data has not been altered in any way. Hash functions are mathematical functions often used to ensure integrity

Availability
Protecting services and data so they are available to people and programs when needed. Although not a strict security function, data that is unavailable can be as damaging as data that is deleted or compromised.

Consistency
Making sure systems behave as expected by authorized users. This includes behavior after fixes are installed or being able to restore data after a disaster or outage.

Control
Regulating access to data and systems through policy or access control lists. This includes not only defining policy, but having sound methods for authorizing users.

Audit
Auditing is an area of increasing importance to the enterprise. What was once part of a best practice is now mandated by legislation such as the Sarbanes-Oxley law. Applying such legislation to computer records and practices can be a challenging endeavor, because the people having the skill to understand the law and audit it are not experts on how to implement it in a computer system, and vice versa.

---

[1] See *Practical UNIX and Internet Security, Second Edition* by Garfinkel, et al.
(http://www.orieilly.com/catalog/puis)

Confidentiality   Protecting information from being read or copied by anyone without express permission. This is also referred to as "privacy".

Most security solutions address one or more of these types of security. This is why they often need to be used in combination to achieve a desired level of confidence that a computer system will be able to operate in a predictable manner and protect the data contained on it or under its control.

1.6 Purpose of security: establishing trust

Fundamentally, security mechanisms are a way for various objects in the operating system to establish trust. Trust is a level of confidence that a system will behave as expected. Absolute security is not achievable, but mechanisms are used to obtain a level of trust.

Trust is needed to ensure that users and programs are authenticated and authorized to perform the function they are about to execute. Trust is also needed to ensure that the system is in a known state, and that the software that is loaded is unaltered. Systems also need to trust remote users and clients, in order to allow access to the system over networks. Even though trust is given to remote users or clients, they must still be controlled to prevent accidental or malicious damage. Trust is also needed to determine what information leaves the computer or corporate network, and those users and processes accessing the data have proper authorization to read it and write it.

In addition to establishing trust, security of AIX V6 is enhanced by preventing misuse of assets. Misuse can be intentional or inadvertent. Many computer vulnerabilities exist because systems are misconfigured or default settings and passwords are still in place. AIX provides many mechanisms for preventing misconfigurations and reduce the availability of malicious code to compromise systems.These services were initially released in AIX 5L V5.3 and have been enhanced in AIX V6.

1.7 Overview of security enhancements to AIX V6 for establishing trust

We have listed the major security enhancements included in AIX V6 below. These features are shipped as part of the operating system and are available to customers free of charge. Because some filesets rely on underlying cryptographic toolkits that are not allowed to be exported with the operating system by United States export regulations, encryption toolkits may be required to be installed separately.

> **Note:** Due to export control restrictions, related cryptographic modules are shipped separately. To use some of these features, those cryptographic modules would need to be installed.

► Enhancements to AIX Security Expert, a security hardening tool.

► Secure By Default (SbD).

► File Permission Manager (FPM)

► A fully implemented enhanced Role Based Access Control system (RBAC)

► Encrypted File Systems (EFS)

► Workload Partitions (WPAR)

► Trusted Execution (TE)

► Trusted AIX

► Secure FTP

► Long Passphrase Support

Table 1-2 shows the new features and a brief description of the type of customers who would want to use those features.

Table 1-2   Security enhancements in AIX V6

Feature	Who should use it
AIX Security Expert	Everyone should use this capability. The customer's hardening scripts can be added as additional rules and invoked by an AIX Security Expert.
Secure By Default	Users who want a minimal number of filesets installed and want strict control on what is added to the base operating system. Network connectivity is disabled in a Secure by Default installation. Also applicable for systems installed on untrusted networks.
File Permission Manager	Customers who are concerned about minimizing programs with setuid bits on.
Role Based Access Control	Customers who have multiple administrators who do not want everyone to have root access and root abilities. This is also useful for administrators from outside companies that must manage servers. RBAC is enabled by default on an AIX V6 installation.
Encrypted File Systems	Customers who want extra protection thoroughly privacy and access achieved by encrypting files, and customers who are required by legislation or business practices to encrypt sensitive data. Requires the installation of a cryptographic toolkit.
Trusted Execution	Customers wanting integrity checking on customer-specific files can add their files to the Trusted Signature Database. AIX operating system files are automatically entered into the TSD. This is a replacement for the Trusted Computing Database.

Feature	Who should use it
Workload Partitions	Customers who want to have added system isolation for applications, to prevent applications from interfering with each other. This allows a sandbox environment where application code can be separated from other code that may offer opportunities for vulnerabilities to be exploited.
Trusted AIX	Customers whose data is of different security classification levels, and where it is imperative that data is not leaked from one level to another. Also for customers who want an mandatory access control security schema where security policies are defined in a top-down hierarchical manner who want to remove the concept of an all -powerful root ID.
Secure FTP	Customers needing to authenticate users using an FTP server for file transfers who do not want to mandate the use of SSH. Requires the creation of an OpenSSL keystore.
Long Passphrase Support	Customers wanting to use longer passwords. No separate installation of filesets is required.

1.8 Trusting the configuration of the OS with AIX Security Expert

Prior to the release of AIX Security Expert in AIX 5L V5.3 TL5, security settings for AIX were distributed among a variety of system and network commands and separate SMIT panels. Creating proper default settings required expertise in many areas, and was a painstaking process. Many administrators ended up hardening their systems after installation by running home-grown scripts that evolved over time. It was difficult to ensure the compliance to settings created in these scripts. Also, if changes to the settings were made that caused undesirable behavior, there was not a good way to restore the original settings.

AIX Security Expert addresses these concerns by building a policy-based set of rules that are implemented by using standard AIX commands. Users can use the graphical interface in WebSM to review a check box style of settings, generally categorized as high, medium, and low security. AIX Security Expert allows security settings to be standardized throughout the enterprise and to be invoked early in the boot process to prevent systems from being vulnerable on the network before customer scripts can be invoked. It also checks for software prerequisites and creates a message if the prerequisites are not satisfied.

AIX will be described in more detail in a later chapter, with example screens. Here we will describe the features and highlights of AIX Security Expert and its enhancements in AIX V6.

1.8.1 AIX Security Expert enhancements for AIX V6

AIX Security Expert has been enhanced with some new features to further improve the security and prevent intrusions. These features include:

1. Invoking Secure by Default for high security setting
2. Centralized Policy Distribution through Lightweight Directory Access Protocol (LDAP)
3. Ability to customize and include user-defined policies
4. Invocation of File Permission Manager command for managing SUID programs
5. More stringent check for weak passwords
6. Performance enhancements for the graphical interface by replacing some Java calls with C code

1.8.2 AIX Security Expert hardening groupings

► AIX Security Expert hardens systems by ensuring the proper configuration based on a desired level of security. One of the more popular features of the tool is its default password rules based on industry best practices. Requiring passwords and password aging are fundamental to securing any system. Table 1-3 shows the security settings AIX Security Experts includes.

Table 1-3   AIX Security Expert hardening groupings

Policy grouping	Description
Password	Aging, length, reset, expiration, and valid chars.
/etc/inetd.conf	Disables many programs, such as tftp, telnet, UDP echo, rshd, and rexd.
/etc/tcpip settings	Disables many TCP apps in high security.
Setuid Policy	Removes setuid bits with FPM.
/etc/inttab	Disables programs like qdaemon, lpd, and piobe. Needs to be explicitly enabled.
Audit Policy	Enables audit if it is not running.
usr/group PW definitions	Passwords must be set and inobvious.
Network Security	Port scan detection, and installs ipsec.
SOX-COBIT configuration assistant	Sets policy to improve record keeping for audits.
Check settings at a later time	Verify that the initial settings are still valid.
Password checking	Stringent dictionary checks for eliminating weak passwords.

1.8.3 AIX Security Expert "undo" option

AIX Security Expert also has the concept of recursive undo. If an administrator finds running an AIX Security expert policy results in an undesired effect because of changed settings, such as the settings being too restrictive to run their environment or application, an "undo" option exists to restore the previous settings. This feature is recursive, and undo can be used multiple times. Care must be taken when changing security settings; not all AIX Security Expert settings can be undone. Table 1-4 shows the settings that cannot be undone.

Table 1-4   Functions that cannot be undone in AIX Security Expert

Check password definitions for High Level Security, Medium Level Security, and Low Level Security.
Enable X-Server access for High Level Security, Medium Level Security, and Low Level Security.
Check user definitions for High Level Security, Medium Level Security, and Low Level Security.
Remove dot from non-root path for High Level Security and AIX Standard Settings.
Check group definitions for High Level Security, Medium Level Security, and Low Level Security.
Remove guest account for High Level Security, Medium Level Security, and Low Level Security.
Changing of login banner.
TCB update for High Level Security, Medium Level Security, and Low Level Security.

1.8.4 Consistency check in AIX Security Expert

AIX Security Expert has the capability to ensure the system is running under the original settings set in the policy file. The `aixpert` command must be run with root authority, and will build a policy file in /etc/security/expert/core/aixpert.xml. This file is used to check the system settings at a later time to ensure the system is still in compliance.

Example 1-1 shows an XML rule for the minimum number of weeks before a password can be changed.

Example 1-1   XML rule for password change

```
<AIXPertEntry name="minagehls">
    <AIXPertRuleType> 1 </<AIXPertRuleType>
    <AIXPertRuleState>Desired</AIXPertRuleState>
    <AIXPertDescription> Specifies the minimum number of weeks to 1 week,
before a password can be changed </AIXPertDescription>
    <AIXPertPrereqList> bos.rte.date, bos.rte.commands, bos.rte.security,
bos.rte.shell, bos.rte.ILS </AIXPertPrereqList>
    <AIXPertCommand>/etc/security/aixexpert/bin/chusrattr</AIXPertCommand>
    <AIXPertArgs>minage= 1 ALL minagehls</AIXPertArgs>
</AIXPertEntry>
```

As you can see from Example 1-1, the policy file contains readable rules that indicate the rule type, the description, the possible prerequisite software, the AIX command to be executed, and the necessary arguments. The AIX Security Expert files can be consulted to learn how AIX commands are used to set certain policies and settings.

1.8.5 Centralized AIX Security Expert policy distribution with LDAP

An important enhancement to AIX Security Expert in AIX V6 is the support for a centralized policy file that is stored in Lightweight Directory Access Protocol (LDAP). Lightweight Directory Access Protocol is a protocol that allows information to be stored centrally in a hierarchical database and can be fetched using the LDAP protocol.

An AIX Security Expert policy file can be created and saved in a central location on an LDAP server. The LDAP server stores the policy file containing the XML rules that is read by AIX Security Expert to determine security settings. Then as other systems in the network need to be hardened, the policy file is fetched from the LDAP server and a consistent policy is distributed and maintained throughout the enterprise.

The different security policies stored on the LDAP server represent the tailored security needs of categorically different systems throughout an organization. For example, a system in the DMZ may require a higher level of security than a back-end test system. The AIX Security Expert GUI can automatically download and graphically present the different security policies stored on the LDAP server. This allows the system administrator to select the best security policy for their environment. A single IT security officer should define and control the policies stored on the LDAP server.

Refer to Chapter 6, "AIX Security Expert" on page 335 for information about how to set ACLs to limit write access to the policy files.

A centrally defined and stored policy file for AIX Security Expert can be used to consistently configure and harden many systems on a network, as shown in Figure 1-1.

Centralized Storage of Security Settings

LDAP server

appliedaixpert.xml
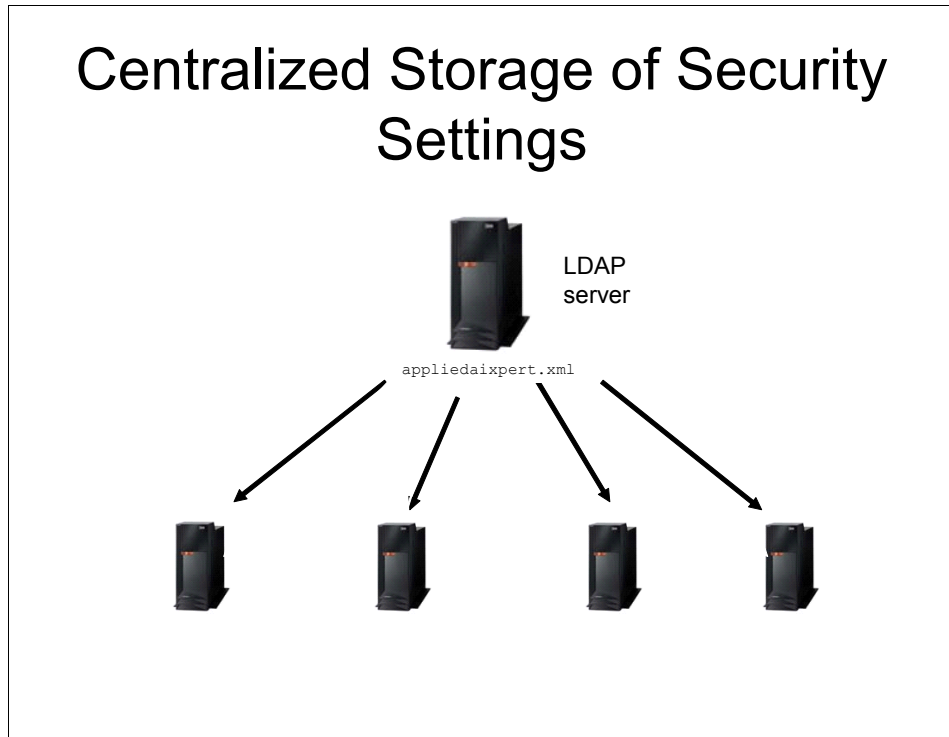
Figure 1-1   Centralized storage of security settings

1.9 Trusting the installation of filesets with Secure by Default

Secure by Default is a new install function, which can be invoked as part of AIX Security Expert to do a minimal installation. The Secure by Default is an even more restricted installation than the high security setting in AIX Security Expert. Secure by Default further protects systems by not allowing network programs to be active before systems are hardened. One principle of security is the fundamental rule of "that which is not explicitly permitted is denied." Secure by Default significantly reduces network vulnerabilities and vulnerabilities from problems in software that can be exploited, by performing a minimal install without network programs. After the initial installation, the administrator only installs what is explicitly necessary on a system.

The customer who would use this function is someone who prefers a bottoms up approach to security where they have the minimal number of filesets and networking function installed, and would enable each specific fileset that is desired. Secure by Default works by removing TCP client applications. Refer to 6.3, "Secure by Default (SbD)" on page 339.

Figure 1-2 shows the traditional installation approach versus the Secure by Default approach.
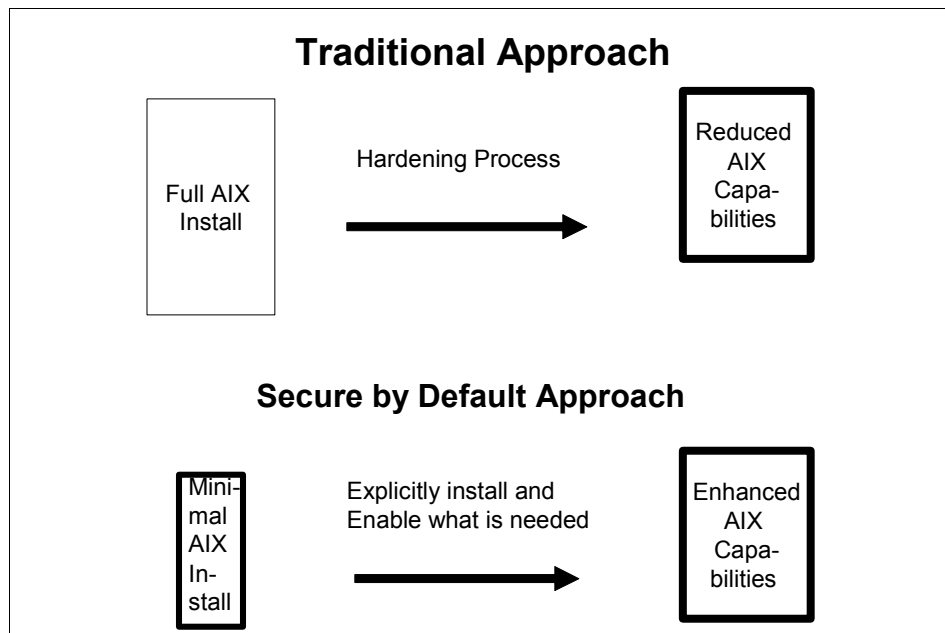
Figure 1-2   Traditional installation approach versus Secure by Default approach

1.10 Trusting system access with File Permission Manager

Reducing the number of set UID bit programs is a best practice for hardening systems. Since AIX has been in existence for a long time and the behavior of the operating system must be stable and predictable for our existing customer base, the set UID bits on many system programs had not been modified. Many of our customers have been writing their own scripts to reduce the number of set UID and set GUID programs. It became clear that what was once a concern to a few security specialists is now a common concern of most customers. As a result, a command was designed to aid customers in reducing the number of set UID bit programs.

File Permission Manager, or `fpm`, is the new command available in AIX to reduce the number of set UID bit programs in AIX. A thorough investigation of each program and a determination of which ones really in fact require the bits was made. Each command affected by `fpm` was analyzed to determine if it was commonly used by a privileged system administrator or if it was needed by the average user. If in general the command was only used by administrators, then `fpm` can reduce the security risk of the set UID permissions by removing these permissions. The programs were divided into different categories, and it was determined which ones would be needed in a high, medium, or low security environment.

The command can be run at different levels with a flag to indicate the security levels of high, medium, or low. Regarding programs that get the s-bit removed, the process is more stringent at the high setting and less stringent with the low setting. The list of bits removed is stored in /usr/lib/security/fpm/data/*xxx*_fpm_data, where xxx is either high, medium, or low. For example, a high security environment would run:

```
fpm -l high
```

Default AIX settings can be restored if the resulting behavior is not what was expected. The default AIX setting can be restored by running:

```
fpm -l default
```

The majority of the programs that have set UID bits removed in the high security setting are removed as well in the medium and low settings. To determine the exact programs where the bits were preserved in the medium settings, one would compare the list in /usr/lib/security/fpm/data/high_fpm_data with /usr/lib/security/fpm/data/med_fpm_data. In order to create a custom list of files with set UID bits removed, one would use the /usr/lib/security/rpm/custom/med_fpm_data to create the custom settings.

One might note that there is only high and medium list files in the data directory. The `fpm -l med` command will disable both the set UID bits and the set GID bits in the /usr/lib/security/fpm/data/med_fpm_data file. In the low setting, only the set GID bits are removed.

Care must be taken when using the `fpm` command, because it may alter the behavior of the system in sometimes adverse ways. If a program or script is depending on using a command with the bits turned on, it may fail with the set uid bits disabled. Therefore, thorough testing should be done when using the fpm command to change the s-bit settings. For more details on using `fpm`, see 3.11.5, "Using the FPM command to reduce SetUID programs" on page 224 and 6.6, "File Permission Manager for managing setuid and setgid programs" on page 357.

1.11 Trusting executables with Trusted Execution

Trusted execution includes a group of features that verify the integrity of files in AIX. Trusted Execution provides for integrity checking of the operating system. At any point in time an administrator can verify the state of the system by comparing the attributes of the important files in the system against a reference database. Additionally, Trusted Execution provides for monitoring of executables and kernel extensions during the load time. It can thus block any attempts to execute malicious code that is not part of the trusted database.

AIX automatically marks certain types of files as trusted, and their signatures will be calculated during installation. Hashes are calculated for runtime checks using the SHA256 hashing algorithm. SHA256 is the Secured Hash Algorithm with a 256-bit key. Additional trusted programs can be identified by the administrator as trusted programs. Those executables are signed and their signature value entered into the Trusted Signature Database (TSD). The values in the TSD are checked by the loader, by calculating the hash value of module and comparing it with the expected value stored in the database. Because even one bit change in a file will radically alter the hash value for that file, a hash is an effective mechanism to ensure the integrity of the file. Executables that do not pass the signature comparison are not permitted to load.

Trusted execution is used to prevent Trojan horses, rootkits, or other methods of tampering with important system files. Bogus copies of files or altered files will fail the integrity check. The types of files that would be appropriate to keep in the TSD include:

► Kernels and kernel extensions

► All setuid root programs

► All setgid root programs

► Any program exclusively run by root or by a member of the system group

► Any program the must be run by the administrator while on the trusted communications path

► Important configuration files

► Any program that may alter system configuration files

The AIX files of the above type are populated in the TSD database by default. Customer-created files of the above type should be added to the Trusted Signature Database to enable the integrity check of each customers' key files.

Trusted execution is run in three different situations:

1. During initial fileset installation. The build process calculates signatures and puts them in the information built in the file set. Selected key files have signatures associated with them, and not all files in the fileset.

2. In "offline" mode, where the customer can use a **cron** job to re-verify the signatures on a periodic basis to ensure files have not been altered. The system administrator can run the integrity verification tool at any time to make sure that the system state has not been modified.

3. During runtime, when certain trusted files are loaded.

During runtime, the loader checks the hash value of the file in the TSD, and calculates the hash of the file. If they match, the file loads. If not, the load will fail. Therefore, if an attacker was successful in loading a tampered version of a system file or other executable file tracked in the TSD, the file would not be able to load and run, and any damage from the compromised file is prevented. These are very key elements in creating a trusted computing environment.

The Trusted Signature Database is organized as a stanza file. A file can be added to the TSD by using the **trustchk -a** command. Once the entries in the database are complete, the TSD can be put in a "locked-down" mode. In this mode, no modifications can be made to the table. In order to take the system out of this mode, a reboot is necessary.

A policy can be created to broadly check defined classes of files, such as kernel extensions, scripts, and executables. This is accomplished by using the `trustchk` command with certain flags, such as CHKEXEC, CHKSCRIPTS, or CHKKERNEXT. If the flag STOP_UNTRUSTED is used, only programs in the TSD will be allowed to run.

For a detailed discussion of Trusted Execution, see Chapter 4, "Trusted Execution environment" on page 251.

1.12 Delegating trust for users and the processes with Role Based Access Control

The traditional access control model in UNIX systems is known as Discretionary Access Control (DAC). This means that the person owning the file or directory has the discretion of setting the permissions for who may read, write, or execute the file. This is the standard rwx to which all UNIX users are accustomed.

In order to perform privileged operations, users must either be part of the system group, or elevate their privileges to root authority. Allowing many system administrators access to the root password of a system or group of systems can be an uncomfortable situation in today's businesses. Responsibilities for various parts of the computer resources and networks is often divided among several administrators. In fact, in some cases administrators may be outside vendors who administer a select application or piece of hardware. Although it requires some planning and policy creation, using the fine-grained access control implementation methods provided by RBAC can give you a lower risk strategy than an alternative strategies where root passwords are distributed to several people.

> If role based access controls are set up properly to define roles and privileges, then the **sudo** command becomes obsolete.

There are additional situations where it may be desirable to divide authorities among two or more people to require multiple people to perform certain operations. This prevents abuse of power, such as creating user accounts and setting passwords, which can be misused to damage company assets or important files, such as audit logs. Currently, the root user ID 0 is needed to perform privileged operations, so root passwords must be shared among many administrators. This can be a logistical headache when passwords need to be updated or administrators leave the company and it is not secure to have non-employees know root passwords.

One popular workaround for tracking users switching to root to perform administrative duties has been the **sudo** command. Using **sudo** still relies on underlying setuid bits on commands. In addition, **sudo** has to be used for the tracking to occur. In most cases, if someone wants to do something undesirable to the system, and they had the root password, they would probably not use the **sudo** command, in order to prevent the audit trail from being created. If role based access controls are set up properly to define roles and privileges, the the **sudo** command becomes obsolete.

1.12.1 AIX V6 Enhanced RBAC compared to AIX RBAC prior to AIX V6

Prior to AIX V6, AIX had some common administrative authorizations defined to do common administrative functions, such as backup, restore, diagnostics, useradmin, groupadmin, passwdadmin, passwdmanage, useraudit, and roleadmin. These roles were always enabled and available for use either by the command-line interface or through SMIT. Although useful, they were limited in scope and did not allow the application of roles to executable code. They required membership in certain groups and often SUID bits or root access was still required to complete most administrative tasks. This early RBAC support is referred to as Legacy RBAC support in AIX (see Figure).
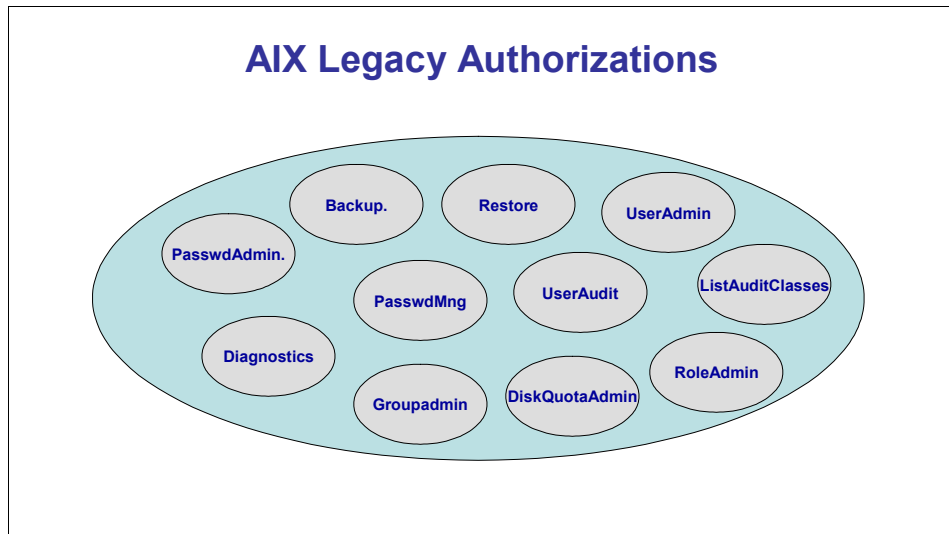
Figure 1-3   AIX legacy authorizations defined prior to AIX V6

There is a system environment variable that indicates if the earlier roles or the AIX V6 enhanced roles are being used. The setting can be changed, but requires a reboot to take effect. The new AIX V6 RBAC support value is named *enhanced_RBAC*. The setting for enhanced_RBAC=TRUE is the default setting for an AIX V6 system.

The new RBAC support in AIX V6 defines authorizations and roles that can be assigned to users. Privileges can be assigned to processes to give them certain attributes so they can gain certain security privileges. These assigned privileges allow the process to bypass privileged operation restrictions.

Enhanced RBAC is enabled on a default installation of AIX V6. RBAC mode in the operating system can be retrieved by applications using the relevant APIs. Refer to Chapter 3, "Role Based Access Control" on page 165 for more details. Additionally, note that Workload Partitions (WPARs) require that AIX be Operating in Enhanced RBAC mode.

Authorizations may be system-defined or user-defined. There are many system defined authorizations that will satisfy many customer's needs. If more authorizations are required, users may create new authorizations and assign them to a new role or one of the existing roles.

To list the system-defined authorizations, enter the command:

```
lsauth -f -a description ALL_SYS
```

1.12.2 Advantages of AIX V6 Role Based Access Control

There are many advantages of the AIX V6 Role Based Access Control mechanisms. They include:

► Selective assignment of privileged access roles to system users

► More levels of granularity than previously offered

► Integration into centralized policy infrastructure with LDAP

► Option to eliminate the root user

► Re-authentication necessary when switching roles, which reduces collaborative attacks

► Ability to define custom authorizations and roles without reprogramming applications

1.12.3 Relationship of authorizations, roles, and privileges

In the new version of RBAC supported in AIX V6, "authorizations" are low level mechanisms that grant access to protected commands. These authorizations can be collected in a group called a role. The user can be assigned a role, or several roles (see Figure).

Elements of AIX 6 RBAC

- Authorizations
  - Mechanism to grant access to commands or certain functionality.

- Roles
  - A container for authorizations that can be assigned to a user.

- Privileges
  - Process attribute that allows process to bypass a security restriction.
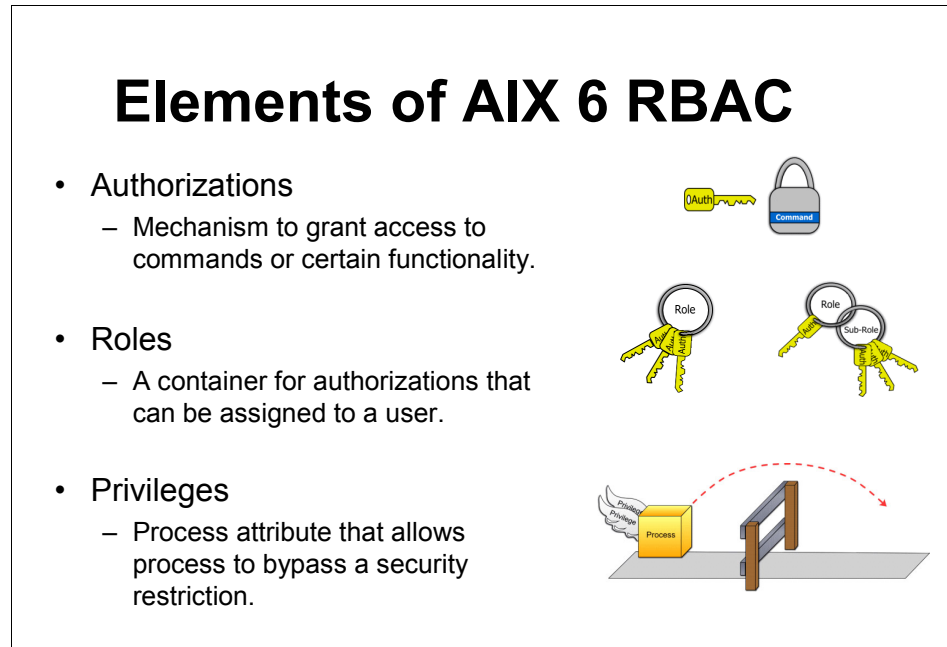
Figure 1-4   Relationship of RBAC authorizations, roles, and privileges

Authorizations can be viewed as keys that give access to privileged operations or commands.

Authorizations are assigned to roles, and roles can be assigned to a particular user. Users may have several roles. Roles are invoked for the time they are needed. Users switch into and out of roles to perform certain privileged operations. Although a user may be allowed to perform many roles, they will assume one role at a time.

As with the distribution of root authority and passwords, assignment of roles should be done with care and only to trusted users. Roles should be assigned for the duration of time that user needs the role, and users should be diligently removed with they leave the job or company.

1.12.4 Privileges versus authorizations

Privileges assign special powers to processes to allow them to perform privileged operations. The ability to assign specific privileges to processes allow a computer system to use fine-grained controls and employ the practice of least privilege. This allows the administrators to mitigate the threat of the over-privileged administrator. Destruction to computers or files can be through careless or malicious acts. The principle of least privilege is the practice where users have the least privilege needed to perform their tasks. Using this approach when configuring administration privileges can help limit possible damage by reducing the scope of a person's access to important files and directories.

Some typical operations that require elevated privileges include rebooting the system, installing software, stopping and audit process, backing up files or creating/removing users.

Figure gives a representation of users having many roles.
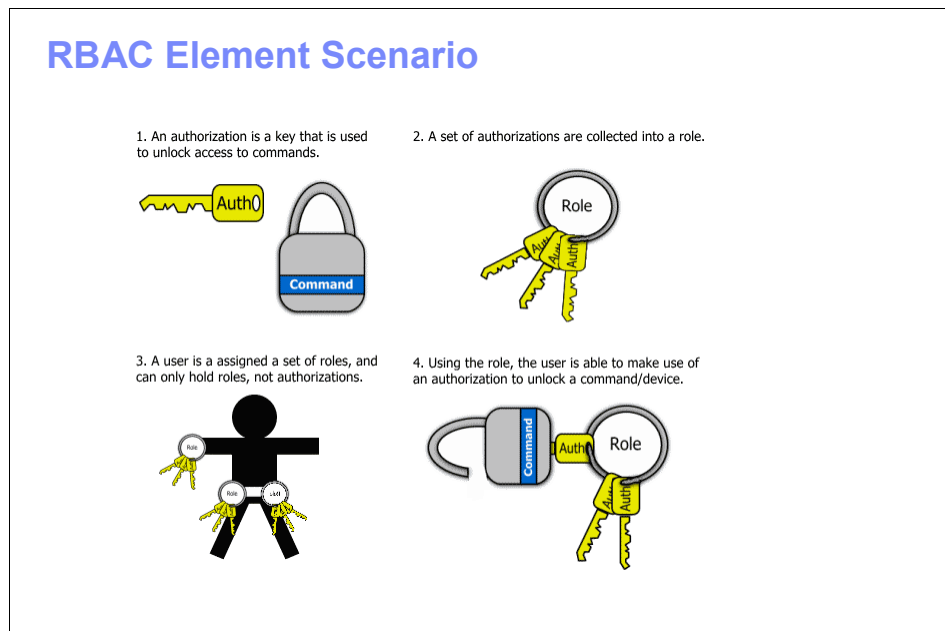
Figure 1-5   Representation of users having many roles

Using privileges assigned to processes to enable privileged commands removes many common attacks that depend on exploiting elevated privileges of a process. Since the user brings with them the authorizations that are defined by the roles they have, they cannot simply inherit the process' privileges.

AIX also has the concept of role sessions. When a user performs a `swrole` command to switch roles, they must re-authenticate to enable the new role. The new authorizations are active while the user is in that role. Once they switch out of the role, their authorizations are removed. This makes it more difficult for credentials to be hijacked, because re-authentication is required and the authorizations are short-lived. Care must be taken when assigning roles to users. Users should only be assigned the roles they need. If a user is no longer in that role, their IDs need to be modified to remove the obsolete role.

Figure shows the changing of role IDs during role session transitions.

.
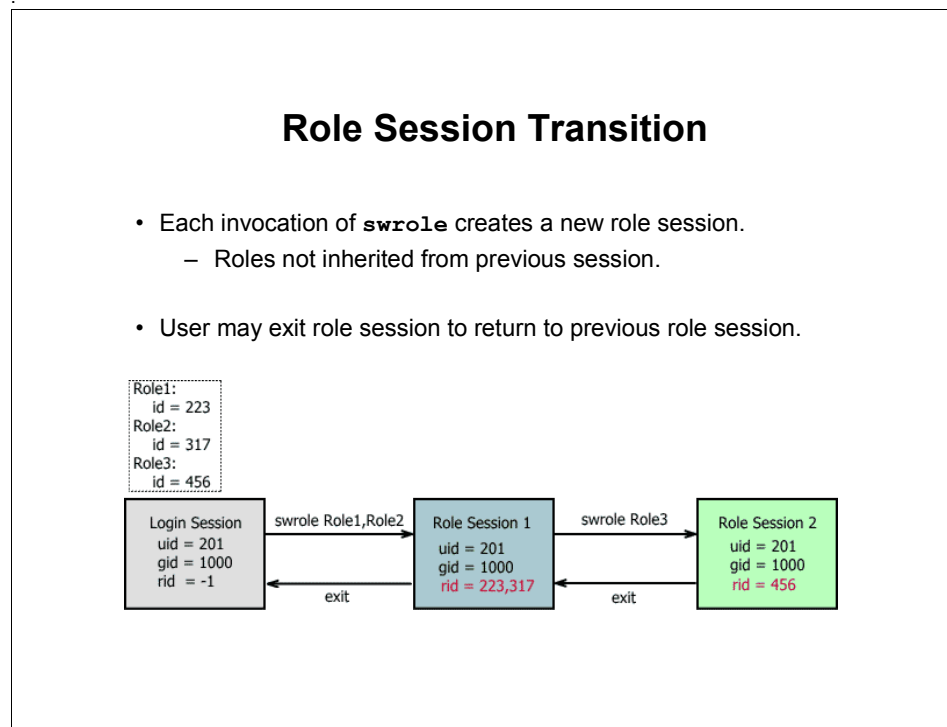
Figure 1-6   Changing of role IDs during role session transitions

As with assigning roles to users, care must be taken when assigning privileges to trusted programs. Privileges should only be granted when and where necessary for the operation of the application. The minimum set of privileges should be granted to allow the program to run properly.

AIX V6 predefines many authorizations in a hierarchical fashion. The AIX RBAC authorizations are enabled by default. Infrastructure to create and track the various authorizations, roles, and privileges are stored in separate RBAC databases. These include:

▶ Authorization database

▶ Role database

▶ Privileged command database

▶ Privileged device database

▶ Privileged file database

Authorizations that are system defined start with "aix". They use a dotted format to show each level. There are aix-specific defined authorizations.

Figure 1-7 shows pictorial view of the AIX hierarchy of aix.system authorizations.
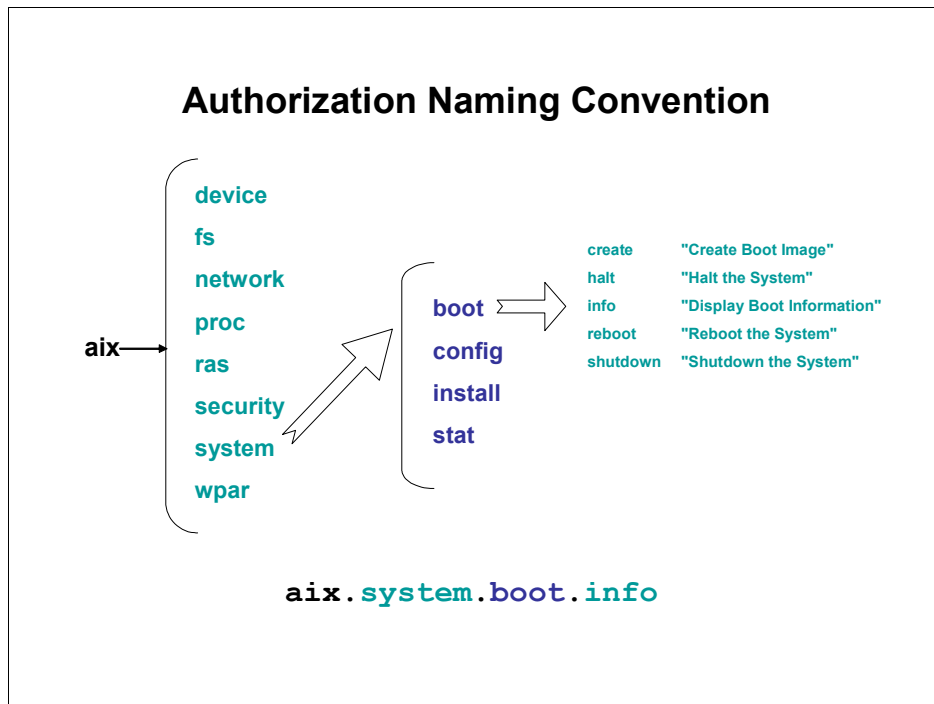
Figure 1-7   Authorization naming convention

To define user-defined authorizations, a similar hierarchy would be created, with the top level being something other than "aix". "aix" at the top level is a reserved word. There is a maximum of nine levels that can be defined in the hierarchy.

Users may define their own hierachy of authorizations and assign them to roles. The base name of the hierachy needs to be something other than aix. For more information about Enhanced Role Based Access Control, please see Chapter 3, "Role Based Access Control" on page 165.

1.13 Trusting file access and providing privacy of files with AIX Encrypted File Systems

AIX V6 includes the capability to encrypt files in the J2 file system with the use of the Encrypted File System (EFS). The encrypting file system encrypts files on a per file basis. Users can either create their own key store, or use a group key store of which they are a member.

The management of user key stores is integrated into the existing user administrator keys. The cryptographic access to files is also integrated into existing commands like `chmod`. This minimized the system administration tasks, and reduces changes of misconfiguration errors. A few EFS specific commands have been added to allow for complete control by the system administrator.

For most operations, the use of an encrypted file is transparent to the user. The underlying commands encrypt and decrypt the data. The specific key and algorithm information for each file is saved in the file meta data, and the commands are programmed to know how to process the files. The keys used to encrypt the files are protected by an asymmetric private key.

The user keystore to access the encrypted file system is opened and loaded upon successful login. A process needing to access an encrypted file verifies the user's credentials and then is able to decrypt the file.

Only a few new commands are needed to use the Encrypted File System. The commands

▶ `efskeymgr` is used to manage and administer the keys.

▶ `efsmgr` is used to manage the encryption of files, directories and file systems.

The Encrypted File System has the following advantages over other encrypted file systems:

► Transparent to users and system administrators.

► Increased file level encryption granularity: Many other encryption products use volume level encryption. All users using files in the volume use the same encryption keys. This type of encryption is useful for protecting the theft of a disk, but does not protect data from being read by others allowed to use the same volume.

► EFS has a unique mode that can protect against a compromised or malicious root user.

► Can assign users to groups and use group keys.

► Centralized keystore.

► Employs AES symmetric encryption algorithm. AES Key length and mode selectable by user.

► EFS is integrated into user administration commands.

1.13.1 Symmetric versus asymmetric encryption

Encrypted File Systems employ both symmetric and asymmetric encryption techniques.

Symmetric ciphers use the same key to encrypt and decrypt. Some examples of symmetric ciphers include DES and AES. Symmetric ciphers are faster, and are commonly used for encrypting "bulk data". They are the ciphers used to encrypt the bulk of private data sent over secure networks, or data that is stored on disk. The main problem with symmetric ciphers is that they require the same key to be used for both encryption and decryption operations, so the keys must be distributed securely to the party performing the decryption. Therefore, the combination of symmetric and asymmetric encryption techniques keys are often used, where the asymmetric keys are used to protect the symmetric keys.

Figure shows symmetric encryption.

Symmetric Encryption

Key   Key

Plaintext   ┌──────────┐ Ciphertext   ┌──────────┐   Original
─────────▶   │Encryption│ ────────────▶ │Decryption│   Plaintext
                 └──────────┘   └──────────┘ ─────────▶
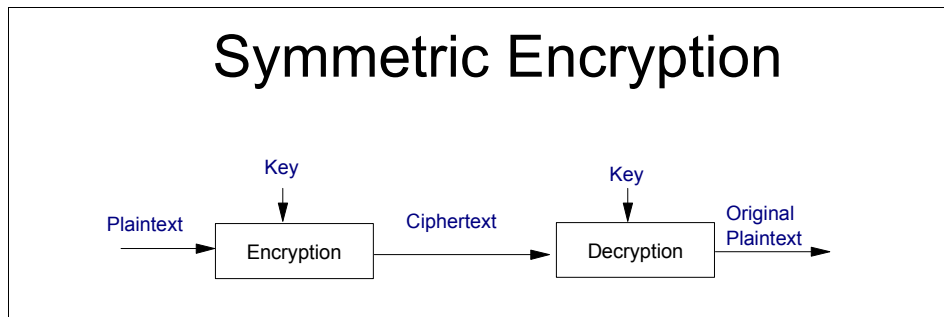
Figure 1-8   Diagram of symmetric encryption

Asymmetric ciphers use key pairs that are mathematically related, called private and public keys. Asymmetric ciphers are a significant invention because the public key, can be distributed safely to others. The private key is used by the owner, and must be stored securely. Typically the owner of the private key encrypts data with their private key, and the receiver or reader of the data decrypts with the public key. There is no risk in distributing the public key in the clear, and it can often be distributed with the encrypted data.

In Figure , P is the plaintext, C is ciphertext (encrypted data), E is the encryption key, and D is the decryption key. In the symmetric case, E=D. In asymmetric, E!=D.

Algorithms

- **Different types of ciphers determined by the key relationship**

- **Symmetric - Encryption and Decryption use Same key**

$$E_k(P)=C$$
$$D_k(C)=P$$
$$D_k(E_k(P))=P$$

- **Asymmetric - Encryption and Decryption use Different Keys which are mathematically related**

$$E_{k1}(P)=C$$
$$D_{k2}(C)=P$$
$$D_{k2}(E_{k1}(P))=P$$
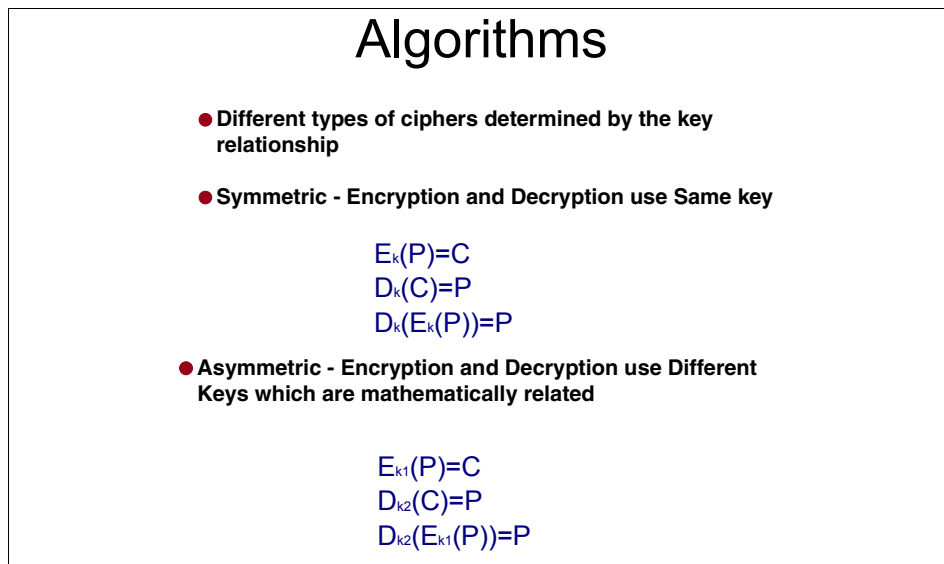
Figure 1-9   Symmetric versus asymmetric algorithms

Data protected by asymmetric algorithms is very secure because the message cannot be guessed by having a sample of the encrypted data or having the public key. The key strength is very large, but the operations are computationally intensive and take a long time. Therefore, public key cryptography is used for encrypting small amounts of important data, or for verifying that a remote party is authentic by their ability to use their private key to decrypt data. In Encrypted File System, asymmetric keys known as RSA keys are used to encrypt the symmetric keys that encrypt the files.

Figure shows the types of encryption algorithms.
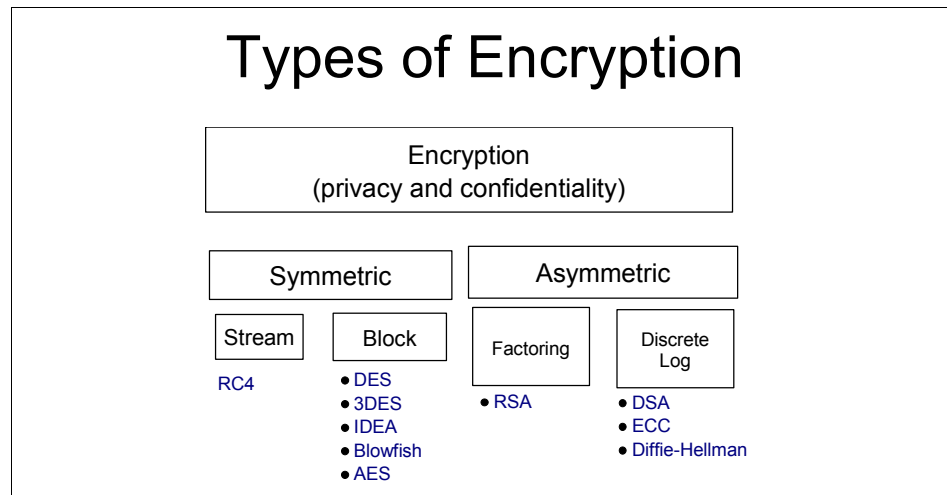
Figure 1-10   Types of encryption algorithms

1.13.2 Advanced Encryption Standard (AES)

Advanced Encryption Algorithm is a cipher that was selected by the National Institute for Standards and Technology (NIST).

For many years, Data Encryption Standard (DES) was the commonly used encryption algorithm. It has a 56-bit key and does bit operations to encipher data. It was adopted by the American National Standards Institute (ANSI) as a standard encryption algorithm in 1981. DES was sufficient to encrypt data literally for decades, and is the algorithm upon which many other standards are built in the field of retail and banking. But as computer processor speeds improved dramatically, it became possible to launch a brute-force attack on data encrypted by DES. In less than a day, computers working in cooperation could try every possible key to unlock the encrypted data.

The algorithm works by doing 16 rounds of bit operations on each half of a 64bit block, then combining the halves together again. Because the nature of the algorithm is to manipulate individual bits, it is not easily optimized in software.

After DES was "cracked" several years ago, Triple DES was the more secure alternative. It uses three encryption passes, once to encrypt, another to decrypt with a different key, and then a third encryption operation was performed, either with the same key used in the first encryption, or with a third key. The resulting ciphertext was, and still is, impossible to crack with a brute force attack, but the process is very computationally intensive. When comparing ciphers and key strengths, one more bit of key length is an exponential increase in the possible keys. Therefore, Triple DES is exponentially many orders of magnitude more secure than DES, but it is three times slower to encrypt and decrypt.

Anticipating the need for a stronger, faster algorithm than DES, the National Institute of Standards and Technology announced an open search for the next cryptographic algorithm in 1996. It was be open to review by the worlds top cryptographers. In November 2001, an algorithm submitted by two Belgium cryptographers, Joan Daemen and Vincent Rijmen, who name it Rijndael (pronounced "rhine-doll"), was selected to become the Advanced Encryption Standard. It had to be secure, fast, and able to be optimized in both software and hardware. AES, as standardized by NIST, is not exactly in the form the Rijndael was originally submitted. It has a fixed block size of 128 bits, and has key sizes of 128, 192, and 512 bits. The original submission allowed for larger block sizes, more choices for key sizes, and more rounds of operations.

AES provides the security of Triple DES with the encryption times of DES or faster. It is used in many encryption products today. As described above, a difference in key length represents an exponential difference in possible keys. For very sensitive, top secret type data, longer key lengths should be used; 128-bit keys are acceptable for most classified type data.

1.13.3 Block versus streaming ciphers

Encryption algorithms are designed to work on either a block of data, or on one byte of data at a time. Ciphers such as DES and AES operate on a block of data. DES used 64-bit blocks and AES uses 128-bit blocks. In some ciphers, such as the original Rijndael cipher in which AES is based, the block size is selectable.

Streaming ciphers encrypt data a byte at a time, and are good for real-time applications, such as streaming voice or video. The most common streaming cipher is RC4. In EFS, AES is a block cipher. Data is fetched from the file in blocks, and it is encrypted or decrypted in 128-bit blocks.

1.13.4 CBC mode versus ECB mode

In EFS, AES can be implemented in two modes called Cipher Block Chaining (CBC) or Electronic Code Book (ECB) mode. Electronic Code Book is the original method in which a certain byte will always encrypt to the same value. It can be viewed as a substituting method whereby the encryption algorithm will always produce the same result for the same plaintext. This can be seen as undesirable, because parts of data often can be predictable, and then the data can be cracked. For example, if a banner is well-known, the encrypted banner data can be used to help determine the relationship between plaintext and ciphertext, and can be used to guess other encrypted messages.

The CBC mode was developed to provide more randomness in the resulting ciphertext. With CBC mode, an initial vector, known as an IV, is used to start the encryption process. The IV is usually unpredictable, and is used as an input to the algorithm for generating the block of ciphertext. Therefore, the data before a block will modify the value of ciphertext. For the first block of data, the IV is a selected, non-predictable value. After the first block is encrypted, the IV is usually a part of the previous block and is fed into the algorithm to calculate the ciphertext for the next block.

If the data to be encrypted is fairly random in nature, ECB mode is faster and suitable. If the data is not random, then CBC mode is more secure and would be preferable.

When an algorithm operates in ECB mode, the cleartext is divided into fixed-size blocks. Each block is then encrypted separately using the same encrypting key. For each cleartext block, there is one corresponding ciphertext block.

This mode of operation has a few characteristics:

► If identical blocks of cleartext are encrypted using the same key, the result is identical blocks of ciphertext.

► Each cleartext block is encrypted independently of the other blocks, which means that more than one cleartext block can be encrypted in parallel. If an application requires I/O operations whose size is a multiple integer of the block size, these operations can be performed simultaneously.

► A potential eavesdropper Eve that has access to cleartext blocks and their corresponding ciphertext blocks can create a database with one-to-one mappings of cleartext blocks and encrypted blocks. If the ciphertext blocks appear in a different message, Eve would then be able to decrypt them without knowing the key. This especially for standard parts of the cleartext message such as headers, templates, forms, or footers, and opens the opportunity to statistical attacks.

- ▶ A potential intruder could substitute original cipher blocks with maliciously-designed cipher blocks that would decrypt to a desired cleartext block without knowing the key. The receiver of the message has no way to realize that this happened.

- ▶ Insertion or deletion of a block does not affect the adjacent blocks.

- ▶ If a cleartext block gets some bits corrupted before the encryption takes place, the errors are propagated only to the corresponding ciphertext block.

- ▶ If one or more bits errors from the ciphertext block get corrupted, decryption of that block will affect only its corresponding cleartext block. In other words, if an error occurs in one block, error propagation is confined within the limits of that block only.

When an algorithm operates in CBC mode, the cleartext is divided into fixed-size blocks. Each block is encrypted using the same encrypting key and the ciphertext produced by the encryption of the previous cleartext block. The first block is encrypted using the key and a data structure named initialization vector. In this manner, the encryption of one block depends on the encryption of the previous block. This mode of operation has a few characteristics:

- ▶ If identical cleartexts are encrypted using the same key and the same initialization vectors, the result is identical blocks of ciphertext. However, if a different initialization vector is used, the resulting ciphertext is different.

- ▶ Each ciphertext block is dependent on the value of the previous ciphertext, which is in turn dependent on the value of its previous cipher block and so on. This has impact on applications that perform random I/O operations.

- ▶ Insertion or deletion of a block does affect the subsequent blocks.

- ▶ If a cleartext block gets some bits corrupted before the encryption takes place, the errors generated during encryption are propagated to all subsequent ciphertext blocks. When decryption takes place, the cleartext block will contain the initial erroneous bits.

If one or more bit errors from the ciphertext block get corrupted, decryption of that block affects its corresponding cleartext block. Every single-bit error affects one bit of the original plaintext.

1.13.5 Selecting key length and modes

As described above, in Encrypted File Systems, AES is used for symmetric encryption of the file data. One may choose between different key lengths and algorithm modes. The longer key lengths will result in stronger encryption; 128 bits of key strength for AES is suitable for all but extremely sensitive data. Since CBC is more secure the ECB, AES-128_CBC is the default choice. If users want slightly better performance and the data is fairly random, AES_128_ECB is a good option.

In informal performance tests made in the lab, there was an approximately 3-5%performance increase from ECB to CBC modes of encryption. Similarly, there was about a 5-10% difference in increasing key lengths. Data encrypted with the fasted choice, AES_128_EBC, was approximately 25-30% faster than data encrypted with the most secure choice, AES_256_CBC.

1.13.6 RSA algorithm

The RSA public key algorithm is used by Encrypted File systems to "wrap" the symmetric key used for encrypting files. RSA is named for the three cryptographers who invented the algorithm: Rivest, Shamir, and Aldeman. They are also the founders of a company called RSA, which produces many encryption software products.

Keys that are encrypted by another algorithm are known as "wrapped" keys. A private RSA key is used to "wrap" the AES key. When an AES key is needed by the file system, it is "unwrapped" with the RSA public key. The RSA algorithm, with a 1024-bit key, is the default choice for the asymmetric algorithm. If users desire a longer key length, RSA_2048 or RSA_4096 may be used. It will take longer to do the wrapping and unwrapping, but it is an operation that is done less frequently than the bulk encryption of the file data. Some applications may require longer key sizes. Selecting the algorithm is done by the person setting the security policy, or it is required by the environment on which the data is being manipulated.

1.13.7 Creating the EFS keystore: installation of CLiC library

Before EFS can be configured, the CLiC library must be installed from the expansion pack. It must be shipped on the expansion pack because import and export laws handle encryption technology separately from operating system code, and it is not possible to ship it with the base operating system. AIX and IBM p servers must obtain a license to export encryption, and therefore cannot ship encryption with the base media. This is an inconvenience, but is required as part of US export and international import laws.

Creating the EFS keystore is performed automatically by the `efsenable` command with calls to the Cryptolite Library in the C library.

EFS can also be configured through SMIT panels, with the "Add an Enhanced Journaled File System" menu. There is a choice for "Enable EFS?", whose default value is no. When this value is set to "yes", the encrypted file system and keystore will be created.

1.13.8 EFS key protection modes: Root Admin or Root Guard mode

EFS allows two modes for key store protection. In the normal Root Admin mode, a root user can reset the user and group key store access passwords. In Root Guard mode, root cannot reset group or user passwords for the keystore. There are pros and cons to each approach. The security officer setting the security policy must weigh the pros and cons and decide if they want to allow root to reset the passwords. If Root Guard mode is selected, care must be taken to back up keystore and password information, or else encrypted data may not be recoverable in the event of a person no longer being able to provide the password.

See Chapter 2, "Encrypted File System" on page 59 for more detailed information about using the encrypted file system.

1.14 Trusting the entire system: Trusted AIX

Trusted AIX is the feature that provides the framework for Mandatory Access Controls and Labeled Security. The roots of Mandatory Access Control (MAC) systems come from the Department of Defense Orange book standards developed in 1985. The labeling capability made it possible to categorize resources and subjects based on security labels. This provided the ability to create vertically defined hierarchies as well as horizontal compartmentalization to separate data and prevent unauthorized access. This system eliminates the concept of the all powerful root ID. Security policies are defined for a site-wide or institution-wide basis and cannot be circumvented by local administrators.

Multi-level security associates subjects and objects. Subjects may be processes or users. Objects may include things like devices, network packets, files and segments. Authorities and policies are defined well before data is populated. The concept of Mandatory Access Control is to prevent data leakage. Therefore, a user or process is defined to operate in a range of values. The objects have a certain classification level. Users may write up or read down. For instance, an information gatherer can write a report that is labeled as higher than his or her level. Once the report is written, he or she can no longer access it.

Trusted AIX allows for the setting of two label types, *Sensitivity Labels* and *integrity labels*. Objects are labeled with their level of security. Processes and users operate in a range or sensitivity levels and the objects they access must be defined within the range of security levels they are permitted to access.

Integrity levels provide a scheme for determining how a user or process can modify an object. Even if it is within their range of security label, they can be further restricted whether they can write the object, read the object, or append to the object, for example.

Trusted AIX uses the Role Based Access Control capabilities to assign privileges and authorizations. Trusted AIX's strict definitions, security controls, and integrity levels allows systems to:

► Protect servers from internal and external attackers
► Compartmentalize and secure applications
► Prevent malicious code from damaging systems
► Limit access to administrative (superuser/root) privileges
► Meet or exceed government standards for maximum security

1.14.1 Components of Trusted AIX

There are several major components to the Trusted AIX system. They include:

► *Sensitivity Labels* that allow for a hierarchical separation of data and who and what processes can access them. Mandatory access controls are applied in addition to Discretionary Access Controls. Therefore, both policy checking must succeed in order to access files. Figure 1-11 gives such an example of Sensitivity Labels.
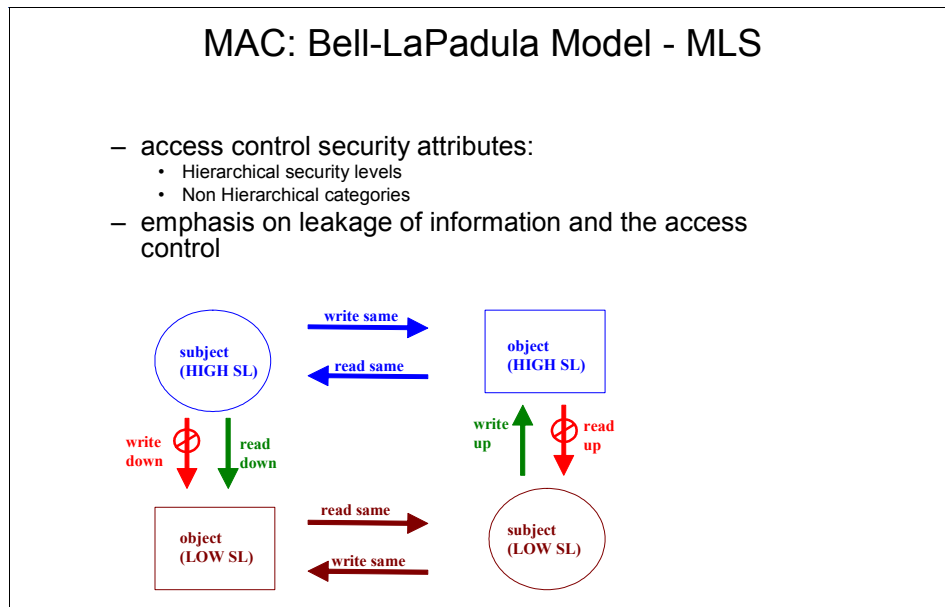
Figure 1-11   Bell-LaPadula Model for controlling access to sensitive data

► *Horizontal classification* based on labeling and compartmentalization.

► *Integrity controls* on how the data can be modified. This determines if data can be read, written, appended, or removed. Figure 1-12 gives an example of an integrity model.
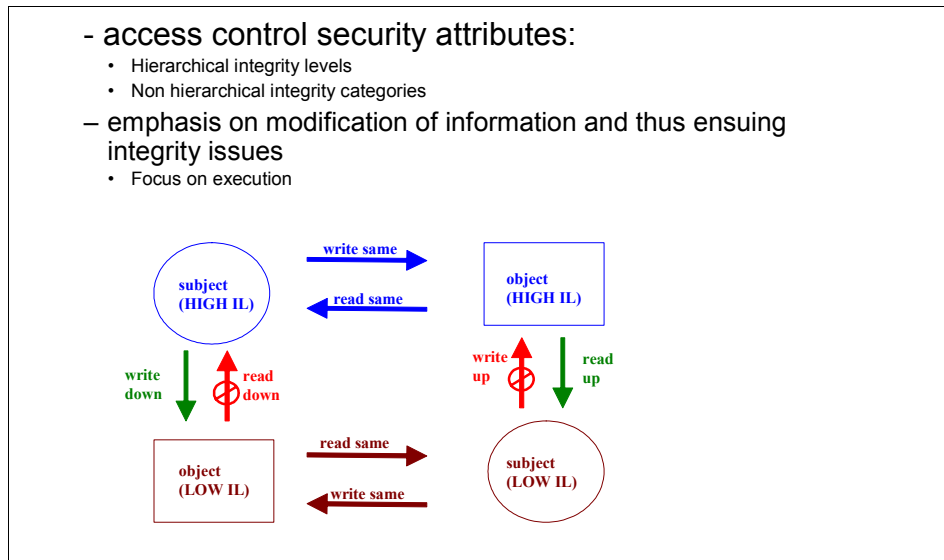
Figure 1-12   Biba integrity model

► *Partitioned directories* allows for finer control of who can view files in a directory. Partitioned directories may be desirable in a shared directory like /tmp, where temporary files can be created in sub-directories that are viewable only to users with the required security label access. This way, those with lower security classification cannot view file names for files with higher security labels, for example.

► *Network controls* using IETF defined Revised Internet Protocol Security Option (RIPSO) headers to label network data and netrules to control the flow of network data into and out of the system. AIX applies network controls to both IPV4 and IPV6 data. *Netrules* are implemented as a combination of host rules and interface rules. Trusted AIX configures default netrule settings. More granular settings may be set by the network administrator.

► *Device access* limits who can mount and unmount devices, and who can access devices

► *Hardening* of the operating system by removing certain components that are vulnerable, such as X traffic.

► Labeling of *printed pages* with the appropriate security classification.

► The use of *Role Based Access Controls* to define authorizations, privileges, and roles to eliminate the concept of root user.

- *Control of Interprocess Communication* (IPC) objects through the use of labels. IPC objects include message queues, semaphores, and shared memory. The IPC objects are extended to include sensitivity and integrity labels.

- *Auditing* of security events. This includes the successful completion of security related actions as well as auditing attempts to violate security policy.

In summary, Trusted AIX is a comprehensive security solution that utilizes Mandatory Access Controls Mandatory Integrity Controls with assigned levels of sensitivity and integrity labeling. Objects are labeled,and users and processes operate within sensitivity ranges.

1.15 In summary: total AIX security capabilities

In addition to describing the new security features that are available with AIX V6.1, we want to present a complete picture of the security features available with AIX. This chart shows the many security features offered with AIX and the release in which they were first made available. Some of the new features for V6.1 are also backported to available for AIX 5L V5.3, such as File Permission Manager and Long Passphrase support. Some features, like Trusted AIX, RBAC, and Trusted execution, were released exclusively on AIX V6.

Note that items listed in a column in the left part of the chart continue to the right. One exception is where Trusted AIX is a replacement product for the vendor product Pitbull Foundation for AIX.

There were some significant security functions that were released in the later releases of V5.3 that should be highlighted because of their customer value. AIX Security Expert was first release in AIX 5L V5.3 TL5, as were TCP wrappers, ipfilters, and stack execution disable.

In Table 1-5, the items in **BOLD** are new features released in AIX 5L V5.3 TL6 and AIX V6.

Table 1-5   New security features

Function	AIX 5L V5.2 and before	AIX 5L V5.3	AIX V6.1
Authorization	► Local Password ► LDAP ► Kerberos	► LDAP Active Directory® enhancements ► **Long Passphrase support** ► **Expanded password algorithm choices***	
Access Control	► Loadable Auth Module ► Limited RBAC	► PAM ► File Permission Manager	► **Fully Implemented RBAC**
Network Security- Confidentiality, integrity, and access control	► IP Security ► Open SSH ► IP Version 6	► TCP wrappers ► IP Security with AES ► ipfilters ► openSSH with Kerberos authentication ► **AIX Security Expert** ► **Secure TCP**	► **Secure FTP** ► **AIX Security Expert enhancement** ► **Secure by Default**
Integrity Checking	► Trusted Computing Base	► Stack Execution Disable	► **Trusted Execution**
File Encryption Confidentiality, access control		► Tape Encryption	► **Encrypted File Systems**
Multi-level Security (Mandatory Access Control)	► Pitbull Foundation for AIX, ► LSPP certification	► Pitbull Foundation for AIX ► LSPP certification	► **Trusted AIX**

Function	AIX 5L V5.2 and before	AIX 5L V5.3	AIX V6.1
System Hardening	AIX CAPP install and certification	► CAPP Install ► AIX Security Expert with high, medium, and low policies, and recursive undo ► **File Protection Manager**	► **AIX Security Expert with SOX compliance assistant, centralized policy, and custom policies** ► **Secure by Default** ► **Trusted AIX**
Encryption Support	4960 coprocessor, 4963 accelerator, and CCA and PKCS11 support	► 4764 accelerator with CCA and PKCS11 support, and Crypto Library in C support with FIPS certification	► **Crypto Library in C support with PKCS11based cryptographic framework**
Auditing Support	AIX Audit framework	► More detailed audit ► AIX Security Expert automatically enables auditing	► **SOX/ COBIT compliance assistance added to AIX Security Expert to enhance capture and reporting capabilities**

1.15.1 LDAP Active Directory enhancements

Support in the `mksecldap` command has been added to seamlessly work with Microsoft®'s Active Directory. Special requirements for working as a client to an Active Directory server have been coded within the `mksecldap` command to make connections to an Active Directory server much simpler. Users should be created on the Microsoft server, and they can be managed by the AIX LDAP client.

In order for Active Directory (AD) to use AIX as a client, it must be UNIX-enabled, and have the UNIX schema installed.

A whitepaper has been written to describe the specifics of connecting to active directory. It is located at:

http://www-128.ibm.com/developerworks/aix/library/au-aixadsupport.html?
ca=dgr-lnxw97AIXclientsupp

Additional support has been added to enable the management of users and groups for RFC2307 schema users. Formerly, this support was limited to users of the aix-schema.

1.15.2 TCP wrappers

TCP wrappers is a program that allows access control on TCP connections. It is a commonly used open source product that has been popular in the UNIX space for years. IP Security provides much higher level of authentication and encryption services, but for some people, TCP wrappers are preferred.

TCP wrappers allow runtime ACL reconfiguration. It can protect traffic, such as FTP, telnet, and r commands. Access from remote hosts can be explicitly permitted or denied. It monitors access to these services and can log either to the AIX log or to its own log file.

1.15.3 IP Security with AES

IP Security has been enhanced for support for the AES encryption algorithm. This provides the equivalent encryption strength of Triple DES with the performance of DES. Key sizes can be selected between 128-, 192-, and 256-bit key lengths.

IP Security provides firewall capability, port scan detection, stateful filtering, encrypted IKE tunnels using authentication with digital certificates, and preshared keys. It supports ipsec tunnels over Network Address Translations addresses (NAT). It also has the ability to shun hosts and do pattern matching on ClamAV antivirus files.

1.15.4 ipfilter support

Ipfilter is an open source filtering tool that is multi-platform. It runs on many versions of UNIX and allows users to filter network packets and perform Network Address Translation. The appeal of ipfilter to customers is that one set of rules can be written and used on a variety of servers in a heterogeneous environment.

1.15.5 Open SSH with Kerberos authentication

AIX Open SSH and PAM support was enhanced in 2006 with support for Kerberos authentication. Future security enhancements will be available in the fall of 2007.

1.15.6 Stack Execution Disable

Stack Execution Disable (SED) is a very key security enhancement that became available with AIX 5L V5.3 TL4. Stack Execution Disable is a feature that prevents code from executing on the stack and heap. Buffer overflows are still the most common mechanism for security exploits.

In Figure on page 48, input is taken into buf that is sized as a 10 character buffer. The strcpy occurs without checking for the input length. If the length of the input string is longer than buf, the variable will overflow on the stack. Attackers use this to insert executable code. This is especially true for open source code, usually network protocol code. Open source gives common protocols where these situations can be found by hackers and exploited. The availability and publicizing of security patches can cause the threats because hackers can often exploit a vulnerability in hours, and people may not patch their systems for days, weeks, or months after a vulnerability is made public. This is the "catch-22 situation" of distributing security patches. With tools on the Web, it is easy to reverse engineer code to find the bug from the patched code, and exploit it. It has been found that exploits are created within hours of a security patch being released, where it may take days, weeks, or months for patches to be installed. Therefore, the patch itself becomes the vehicle for spreading viruses and other threats.

Using a preventative approach such as Stack Execution Disable provides "zero day" protection without any patching or code changes. "Zero Day" means the countermeasure are effective even before the attack is launched. There are zero days where systems are vulnerable.
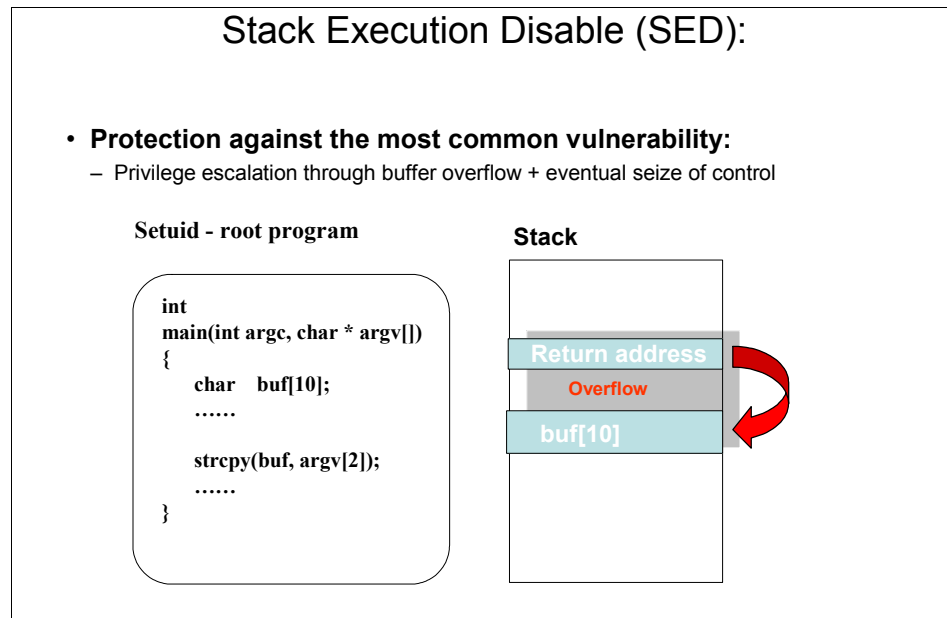
Figure 1-13   Stack Execution Disable prevents buffer overflow attacks

Because Stack Execution Disable will change the behavior of a customer's system, it is not enabled by default. Customers must select it. There are a few types of applications that execute on the stack in normal operating mode, such as gcc and Java. Stack Execution Disable can be run in a monitor mode so that customers can try the feature in their environment and determine if it would trigger in a normal mode.

There is an ability to set the granularity in setting SED to run. It can run in an overall on/off mode, or it can be selected for certain executables, or excluded for certain files. It can also be enabled for setuid and setgid programs. This is a very powerful setting, because escalating of privileges in a overflow of a setuid or setgid program is a common attack profile. POWER4™ or later hardware is required to run this feature.

If the SED feature is triggered by code executing on the stack, the process will be terminated and an error log entry is generated.

Stack Execution Disable and port scan detection

SED can be combined with port scan detection for a more complete security solution. Most attacks are preceded by scanning the computer for listening ports. Using port scan detection will reduce a hacker's access to the system. When inactive connections are scanned, that remote ID is blocked automatically for five minutes. No response is sent back, and the remote host's IP address is saved for denying future packets. A filter rule is automatically dropped into the filter table and the offending host will not receive any response to their probes.

1.15.7 4764 Cryptographic Accelerator with CCA and PKCS11 support

AIX has support for the 4764 Cryptographic Accelerator. It is a PCIX adapter that has self-destructing tamper-detecting packaging that is certified at the highest level - FIPS 140 -2 level 4. FIPS is the Federal Information Processing Standard, which is the government standard developed by NIST. Level 4 is the highest level attainable and has very stringent security requirements. The 4764 has two software application programming interfaces. It runs host code and firmware on the adapter that can perform IBM Common Cryptographic Architecture (CCA) calls as well as the industry standard PKCS11 interface.

CCA is a very robust set of application programming interfaces geared towards the needs of the banking and finance industries. Firms that run ATM machines or major bank servers need to have signing keys stored in tamper-detecting hardware. Keys are typically loaded by a procedure that requires two different administrators that each have a part of the key. Currently, CCA supports DES and Triple DES symmetric algorithms, with RSA 1024 and 2048 key length for key generation, encryption, and signing functions. It includes SHA1 and MD5 hashing mechanisms.

The PKCS11 interface supports DES, Triple DES, and AES symmetric encryption. It also supports SHA1, SHA256, and MD5 hashing.

In addition, the card offers two cryptographic-quality hardware random number generators. The entropy is obtained from a semiconductor junction.

The software is implemented on a Linux base, and it designed to be updated in the field. This allows for future cryptographic enhancements to be accommodated without requiring customers to update their adapters.

1.16 AIX certifications: independent assurance of security functions

IBM has invested much effort and expense in the independent assurance of security functions through the process of obtaining security certifications. The use of independent auditors with deep security skills provides for a thorough review of designs, code, and security testing of AIX. Third party companies are hired to review the documentation and code, write the necessary security documents and respond to comments from the security standards bodies. Once the entire process is complete, the security certification is granted.

1.16.1 Background on security standards

Many of today's security standards have evolved from the Department of Defense "Orange Book" security requirements. The United States had a security standard known as Trusted Computer Security (TC Sec) evaluation criteria. Similarly, European nations had their own standards, known as Information Technology Security (ITSEC) evaluation criteria. These standards were similar, but had differently named levels and protection profiles. This proved to be problematic for customers and developers trying to create products to satisfy the needs for both markets. Soon it was clear that a common standard was necessary.

A new security organization was formed to create an international standard. This organization is called Common Criteria. They converged the two standards into one set of requirements and security level definitions. The standards are internationally recognized. This greatly simplifies the manufacturer's ability to satisfy security requirements for equipment use in many different countries. The Web site for Common Criteria information is available at:

http://www.commoncriteriaportal.org

With the Common Criteria method of certifying products, a manufacturer selects the protection profile they want to use for their security evaluation. Then they select the level at which they will be evaluated. Every profile has the same Evaluation Assurance Levels (EAL) levels, from EAL1 to EAL7.

Then they must determine the target of evaluation (TOE), or basically the scope of the evaluation. This includes the hardware and software being used to satisfy the security requirements. The product definition and the scope of the evaluation is documented in the key foundational document called the "security target."

In the past, AIX has certified its products against what is known as the CAPP and LSPP profiles. A further description of these profiles are available on the Common Criteria Web site at

http://www.commoncriteriaportal.org/public/developer/index.php?menu=8

From the home page, select **Developer** and then **List of PPs**. (A list of protection profiles. All of the various protection profiles are available in PDF format).

1.16.2 Security profiles for AIX V6.1: CAPP, LSPP, and RBACPP

The security functions in AIX V6 will be submitted for certification against the CAPP, LSPP, and RBACPP profiles. The RBACPP profile is the role based access control profile. It will be using the enhanced RBAC functionality and will be a new profile for AIX to certify.

1.16.3 The Controlled Access Protection Profile (CAPP)

The Controlled Access Protection Profile is a general purpose computer system profile that includes many administrative, authentication, and authorization services within the operating system. All system calls are reviewed as well as password rules, lockout mechanism, authentication schemes, and complete code development, build, distribution, fix, and warranty services. Vulnerability programs for fixing bugs are reviewed in depth. During the certification process, documents are submitted describing this different security mechanisms of AIX. Design documents must exist, be approved, and be accurate to the current product behavior. The code is reviewed for security function and possible flaws during vulnerability analysis. Then the actual running system is tested by the auditor at an on-site test visit. If code is developed at more than one location, all locations must be reviewed and visited.

The description of the system under review is documented in the security target. Since the system must undergo strict security review, a reduced version of AIX is submitted for evaluation. The specifics of the functions included are described in the security target document. The security target can be viewed at http://www.commoncriteriaportal.org by selecting the **Evaluated Products** link. Look in the section for operating systems.

1.16.4 Labeled Security Protection Profile (LSPP)

Labeled Security Protection Profile is a profile for multi-level security products that implement Mandatory Access Controls (MAC). This takes the control away from the individual owner and moves it to a central scheme to determine access to data and devices. It is used, for example, for managing confidential, secret, and top secret data. The root user is not able to change the security settings.

With AIX V6, AIX will undergo LSPP evaluation with all native code. The prior MLS features that required third-party software will be part of AIX with Trusted AIX.

1.16.5 Role Based Access Control Protection Profile (RBACPP)

The Role Based Access Control Protection Profile is a profile defined by Common Criteria. This profile is used for evaluating systems that have Role Based Access Control schemes. Traditional system administration functions are mapped to a set of roles that have the authority to make changes to system settings for devices and files within the control of their role.

1.16.6 Current AIX certifications: CAPP and LSPP

AIX has several key security certifications. It has been certified according to the Common Criteria's Controlled Access Protection Profile (CAPP) at level EAL4+. This certification is according to a protection profile defined by the security consortium's Common Criteria. It is an internationally recognized standard. Information on the Common Criteria organization can be viewed at http://www.commoncriteriaportal.org. This organization defines different security profiles, and companies can certify their products at various levels of security. AXI received the CAPP certification for AIX 5L V5.2 TL6 and AIX 5L V5.3 TL5.

AIX and Pitbull Foundation by Argus® Systems is also certified at Labeled Security Protection Profile (LSPP). This is a very strict security profile that requires the use of Mandatory Access Controls and data labeling. Pitbull Foundation is a specially modified version of AIX to implement data labeling that is available for AIX 5L V5.2 and AIX 5L V5.3. In AIX V6.1, the Mandatory Access Control and data labeling function is built into AIX.

At the time of the publishing of this book, AIX V6 is in the process of being certified as well, for CAPP, LSPP and Role Based Protection Profile (RBPP). Please contact the AIX security group with specific questions about the progress of these certifications if they are required for your environment.

AIX V6 is different than prior operating systems where different installs were needed for CAPP and LSPP. In AIX V6, there will be an install option for whether a CAPP system or a LSPP system is desired. The LSPP system builds upon the security included in the CAPP profile, and extends it for the Labeled Security function. Because many environments may require CAPP without the extra security and configuration required of an LSPP system, there is a choice of configuring AIX V6 in CAPP or LSPP mode.

1.16.7 Evaluation and assurance levels for Common Criteria

Common Criteria uses a numbering scheme of its evaluation levels from EAL1 to EAL7. The earlier US standard TC Sec had levels known as C1, C2, and B1. Early versions of AIX V4 were certified to the C2 level, and an early version of a security product known as BestX by Group Bull was certified at B1, which was a labeled security product. Since that time, the Common Criteria standards have come about and now the EAL levels are used. AIX starting using the Common Criteria profiles starting with AIX 5L V5.2. Sometimes people in the field will still refer to the old standards and ask for a C2 level of AIX. They should be directed to the EAL levels of the Common Criteria standards described in Figure .

Evaluation and Assurance Level - EAL		
Common Criteria	US TCSEC	European ITSEC
EAL2	C1: Descretionary Sec Prot.	E1
EAL3	C2: Controlled Access Prot.	E2
EAL4	B1: Labeled Security Protection (LSPP)	E3
EAL5	semiformally designed and tested	E4
EAL6	semiformally verified design and tested	E5
EAL7	formally verified design and tested	E6

Figure 1-14   Relationship of CC levels to TS Sec and IT Sec

1.16.8 What does EAL4+ mean

EAL4+ is the evaluation level for which AIX obtained its security certification. It means "Evaluation Assurance Level 4 Plus". The "plus" refers to an additional requirement known as "flaw remediation". This means that there is a program in place to actively respond to any reported security problems and to be able to notify and distribute security patches to customers very quickly

Table 1-6 is a chart of the different certification levels and what they mean. Most operating systems have code from many sources, and it is difficult to obtain more than EAL4+ without the code being designed and written with the security requirements in mind. In fact, many operating systems are evaluated at EAL3, because design documents are required for security-related code, and this is a huge task to create if the documents do not already exist.

A few operating systems are listed at the EAL5 level. It is very difficult to certify code to the EAL5 level. When an product is assured at EAL5, it is at a significantly more stringent standard. Practically speaking, the code has to be written with strong security principles and architecture in mind, and it is very likely that there will be a situation where the code is written by one company and the security functions were planned from the inception of the product.

Unlike EAL1-EAL4, the EAL5 rating is only acknowledged by the country that certified it, except in the case of Europe, where European countries have agreed to recognize EAL5 from another European county.

Table 1-6   Common criteria EAL levels and their meaning

Evaluation and assurance levels	
EAL1	Functionally tested
EAL2	Structurally tested
EAL3	Methodically tested and reviewed
EAL4	Methodically designed, tested, and reviewed
EAL4+	EAL4 with flaw remediation (AIX is EAL4+.)
EAL5	Semiformally designed and tested
EAL6	Semiformally verified design and tested
EAL7	Formally verified design and tested

1.16.9 Definition of EAL4

Products that are evaluated at EAL4 are methodically designed, tested, and reviewed. EAL4 permits a developer to maximize assurance gained from positive security engineering based on good commercial development practices. Although rigorous, these practices do not require substantial specialist knowledge, skills, and other resources. EAL4 is the highest level at which it is likely to be economically feasible to retrofit to an existing product line. It is applicable in those circumstances where the developers or users require a moderate to high level of independently assured security in conventional commodity Target of Evaluations (TOEs) and there is a willingness to incur some additional security-specific engineering costs.

An EAL4 evaluation provides an analysis supported by the low-level design of the TOE and a subset of the implementation. Testing is supported by an independent search for obvious vulnerabilities. Development controls are supported by a life cycle model, identification of tools, and automated configuration management.

EAL4+ requires the presence of a flaw remediation program. For AIX, there are resources dedicated to monitor and respond to vulnerability reports. This includes reports of other operating systems, protocols, and applications where a similar vulnerability exists in AIX. Security vulnerabilities that are found during internal testing must also be reported and fixes made available. There is a subscription service that exists to distribute information about possible security patches, as shown in Figure 1-15.
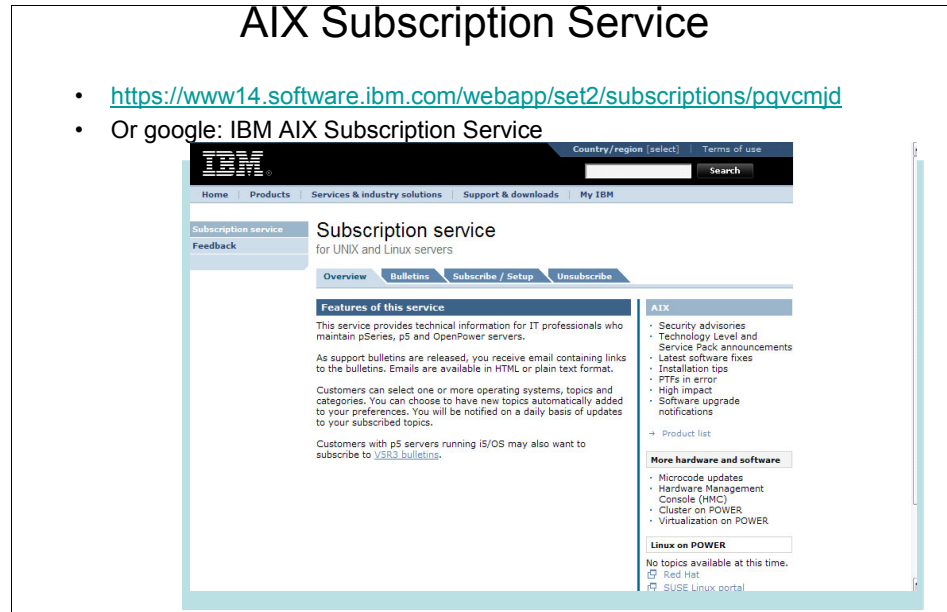
Figure 1-15   AIX Subscription Service for Security information and Fix Distribution

1.16.10 Running a system in CAPP or LSPP mode

In order to run a system in a strict CAPP mode, the exact version of the operating system must be installed. This is an option on the installation menu. The AIX operating system is hardened for CAPP and has some settings specifically set to meet the requirements of the CAPP standard. Therefore, if a customer wants to run a CAPP version of AIX, it must be a fresh install. There is no way to convert an already installed system into an officially CAPP certified version of AIX.

One of the limitations of running a CAPP certified system is that maintenance levels cannot be applied to it. The install tools do not allow a CAPP certified system to be updated.

In some cases, customers feel that the patches that need to be applied are important enough to warrant taking the system out of CAPP mode. If this is true, the system must be taken out of CAPP mode and have the fixes applied.

It is possible to perform most of the same hardening procedures that are performed in AIX after the patches are installed. First, the system must be taken out of Common Criteria mode, or CC Mode. This is accomplished by issuing:

```
> odmchange -o PdAt -q "attribute=TCB_STATE" odm.data
```

Then the fixes are applied, the system is hardened according to the Common Criteria scripts by running:

```
> set_state_to_CC
```

Once the setting is set back to CC, it is not strictly speaking in CC mode, as the system has been modified, but it will behave similarly to a CC configured system.

2

Encrypted File System

2.1 EFS

EFS is designed so that each file is encrypted with a unique key. The cryptographic information is kept in file extended attributes for each file. EFS uses EA (Extended Attributes) Version 2.

Each file is encrypted before being written to disk. The files are decrypted when they are read from the disk into memory, so file data is kept in memory in clear format. The advantage is that the data is decrypted only once. Although the file data is already in memory in the clear format, when another user requires access to the file, his or her security credentials are verified before being granted access to the data. If the user is not entitled to access the file, the access will be refused.

File encryption does not eliminate the role of traditional access permissions, and rather adds much more granularity and flexibility.

EFS internal features have also been integrated into **fdb**.

Traditional commands used for file manipulation such as **mv** or **cp** have had a new flag added in order to support operations with encrypted file systems.

2.2 EFS prerequisites

In order to be able to create and use EFS-enabled file systems on your system, there are a few prerequisites that need to be met:

► CryptoLite in C (CliC) cryptographic library needs to be installed.

► The RBAC has to be enabled.

► You must explicitly enable the system to use the EFS file systems.

2.2.1 CLiC installation

In order to create and use EFS, you must install the CLiC cryptographic fileset at level 430.CLiC comes at no cost and can be found on the Expansion Pack that comes along with the V Base CDs.

After installing the CLiC filesets, you should obtain an output similar to the one shown in Example 2-1.

Example 2-1 Summary of successful installation of CLiC filesets

```
Installation Summary
--------------------
Name                        Level       Part      Event       Result
-------------------------------------------------------------------------------
clic.rte.lib                4.3.0.0     USR       APPLY       SUCCESS
clic.rte.kernext            4.3.0.0     USR       APPLY       SUCCESS
clic.rte.includes           4.3.0.0     USR       APPLY       SUCCESS
clic.rte.kernext            4.3.0.0     ROOT      APPLY       SUCCESS
```

More details about CliC and the way AIX uses it can be found in Appendix A, "Crypto Lib in C (CLiC)" on page 377.

2.2.2 Enabling EFS for file systems

In order to enable the system to use encrypted file systems, you must run the `efsenable` command.

You can enable file system encryption as follows:

► Log in to the system as a user who is entitled to run this command, either root or a member of the group security with the additional *aix.security.efs* authorization.

► Run the `efsenable` command to activate the EFS. If CLiC is not installed, the command fails and displays an error message similar to the output shown in Example 2-2.

► Type the password that is used to protect the initial keystore

Example 2-2 Trying to EFS-enable a system without having installed CLiC

```
# efsenable -a
/usr/lib/drivers/crypto/clickext: A file or directory in the path name
does not exist.
Unable to load CLiC kernel extension. Please check your installation.
```

Important: If the login password and keystore password are the same, the user keystore is opened automatically at login time and user security credentials are associated with user processes. For root, this feature allows for seamless integration of security privileges with all system administration commands.

2.2.3 The efsenable command

Running the `efsenable` command does the following:

► Creates the /var/efs directory. In this directory, the following directories are created:

- The efs_admin directory, which contains the efs_admin keystore and a lock file.

- The users directory: This directory contains a lock file used to get exclusive access to the /var/efs/users directory. For every user defined on the system, a subdirectory having the same name as the user name is created. Each subdirectory contains the corresponding user keystore and a lock file. The lock file is used to get exclusive access to the /var/efs/users/username directory. At the time of installation, a subdirectory, a keystore, and a lock file are created corresponding to the user that ran the `efsenable` command (usually root).

- The groups directory: This directory contains a lock file used to get exclusive access to /var/efs/groups directory. For every group defined on the system, a subdirectory having the same name as the group name is created. Each subdirectory contains the corresponding group keystore and a lock file. The lock file is used to get exclusive access to /var/efs/groups/groupname directory. At the time of installation, a subdirectory, a keystore, and a lock file are created corresponding to the group security.

► Sets the default length for using public/private keys.

► Sets the default key length and mode for the AES algorithm that is used for file encryption.

► Sets the default keystore administration mode.

► Sets the algorithm for EFS administration key.

► Specifies if the user can change the mode for how their keystore is administered.

► Creates a file named /var/efs/efsenabled that shows that EFS has been enabled on the system.

► Updates /etc/security/user and /etc/security/group to contain the new EFS-specific attributes.

► Updates the Config_Rules ODM database.

► Loads the efs kernel extension.

In Example 2-3, we show how to EFS-enable a system as follows:

▶ Log in as the root user.

▶ Running the **efsenable** command.

▶ Verify that efs kernel extension has been loaded in the kernel using the **genkex** command.

▶ Display the structure of files and directories that have been created, as described in 2.2.3, "The efsenable command" on page 62.

Example 2-3 Enabling file system encryption

```
# id
uid=0(root) gid=0(system)
groups=2(bin),3(sys),7(security),8(cron),10(audit),11(lp)
# efsenable -a
Enter password to protect your initial keystore:
Enter the same password again:
#
# genkex -d|grep -i crypto
        4b44000    37748         4b73000     873c
/usr/lib/drivers/crypto/clickext

# ls -al /var/efs
total 8
drwxr-xr-x    5 root     system         256 May 09 11:03 .
drwxr-xr-x   29 bin      bin           4096 May 09 11:03 ..
drwx------    2 root     system         256 May 09 11:03 efs_admin
-rw-r--r--    1 root     system           0 May 09 11:03 efsenabled
drwx------    3 root     system         256 May 09 11:03 groups
drwx------    3 root     system         256 May 09 11:03 users
# ls -al /var/efs/efs_admin
total 8
drwx------    2 root     system         256 May 09 11:03 .
drwxr-xr-x    5 root     system         256 May 09 11:03 ..
-rw-------    1 root     system           0 May 09 11:03 .lock
-rw-------    1 root     system        1709 May 09 11:03 keystore
# ls -al /var/efs/users
total 0
drwx------    3 root     system         256 May 09 11:03 .
drwxr-xr-x    5 root     system         256 May 09 11:03 ..
-rw-------    1 root     system           0 May 09 11:03 .lock
drwx------    2 root     system         256 May 09 11:03 root
# ls -al /var/efs/groups
total 0
drwx------    3 root     system         256 May 09 11:03 .
```

```
drwxr-xr-x    5 root     system           256 May 09 11:03 ..
-rw-------    1 root     system             0 May 09 11:03 .lock
drwx------    2 root     system           256 May 09 11:03 security
# ls -al /var/efs/users/root
total 8
drwx------    2 root     system           256 May 09 11:03 .
drwx------    3 root     system           256 May 09 11:03 ..
-rw-------    1 root     system             0 May 09 11:03 .lock
-rw-------    1 root     system          2128 May 09 11:03 keystore
# ls -al /var/efs/groups/security
total 8
drwx------    2 root     system           256 May 09 11:03 .
drwx------    3 root     system           256 May 09 11:03 ..
-rw-------    1 root     system             0 May 09 11:03 .lock
-rw-------    1 root     system          2062 May 09 11:03 keystore
```

When you want to run the **efsenable** command on a system that had already this
command run, you get the following message:

```
# efsenable -a
EFS has already been enabled on this system.
```

2.2.4 Usage of lock files

Lock files are used to maintain the integrity of user and group keystores during
system operation.

When an operation that modifies a user keystore takes place, the following steps
occur:

► /var/efs/users is locked.

► /var/efs/users/user is locked.

► /var/efs/users is unlocked to allow subsequent access.

► /var/efs/users/user remains locked until the keystore operation completes.

► Keystore manipulation takes place.

► /var/efs/users/user is unlocked.

> **Tip:** Detailed log messages generated during EFS related operations can be
> found in the syslog.out file.

2.3 Managing encrypted file systems and encrypted files

In this section, we describe the main operations that can be performed upon an EFS-enabled file system.

2.3.1 Creating an EFS

We present the interfaces used to create an EFS-enabled file system or to convert an already existing JFS2 file system to an EFS-enabled one.

You can use the SMIT menu to create an encrypted file system, as shown in Example 2-4. When you create a enhanced journaled file system, the option of creating it encrypted is disable by default.

Example 2-4   Creating an encrypted file system using SMIT

```
Add an Enhanced Journaled File System

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                               [Entry Fields]
  Volume group name                             rootvg
  SIZE of file system
      Unit Size                        Megabytes
+
*      Number of units              [100]
#
* MOUNT POINT                                    [/fs1]
  Mount AUTOMATICALLY at system restart?        no
+
  PERMISSIONS                        read/write
+
  Mount OPTIONS                      []
+
  Block Size (bytes)                  4096
+
  Logical Volume for Log
+
  Inline Log size (MBytes)           []
#
  Extended Attribute Format
+
```

```
  ENABLE Quota Management?                       no
+
  Enable EFS?                                   yes
+
  Allow internal snapshots?                      no
+




F1=Help                    F2=Refresh                   F3=Cancel
F4=List
F5=Reset                   F6=Command                   F7=Edit
F8=Image
F9=Shell                   F10=Exit                     Enter=Do
```

You can create an EFS file system also using the command-line interface.

The following command creates an 100 MB encrypted file system that will be mounted under /fs2:

```
crfs -v jfs2 -g rootvg -m /fs2 -a size=100M -a efs=yes
```

You can verify the creation of the stanza corresponding to the newly-created file system using the `lsfs -q` command, as shown in Example 2-5.

Example 2-5   Using the lsfs command to display EFS attributes of the file systems

```
# lsfs -q /efs
Name            Nodename    Mount Pt            VFS   Size    Options
Auto Accounting
/dev/fslv00     --          /efs                jfs2  212992  rw
no   no
  (lv size: 212992, fs size: 212992, block size: 4096, sparse files:
yes, inline log: no, inline log size: 0, EAformat: v2, Quota: no,
DMAPI: no, VIX: no, EFS: yes, ISNAPSHOT: no)
```

Please note that EAformat for the encrypted file systems is Version 2 and the value of the EFS attribute is set to yes.

You can enable the encryption for an already existing JFS2 file system using either SMIT panels or the `chfs` command. You need to set the value of the efs attribute to yes.

This operation automatically changes EA format from Version 1 to Version 2. Data previously stored in EA Version 1 such as ACL is converted to EA Version 2.

At the time of this writing, there is no WebSM interface available for creation of an EFS-enabled file system or for converting a regular JFS2 file system to be EFS-enabled.

Important: File systems /, /usr, /var, and /opt cannot be EFS enabled.

2.3.2 Operations with EFS-enabled file systems

Because the EFS features are completely integrated in the JFS2 file system design, all operations that can be performed on a regular JFS2 file system can be performed on EFS-enabled file systems as well. These operations include:

► Mounting file system

► Unmounting file system

► Increasing the size of the file system

► Decreasing the size of the file system

► Defragmenting the file system

► Removing the file system

Restriction: Due to the fact that file security information is kept in file metadata, the EFS file systems cannot be exported through NFS, and they cannot be locally mounted through NFS.

2.3.3 Encryption inheritance

The concept of encryption inheritance indicates if a file or directory inherits both the property of being encrypted and all encryption parameters from its parent directory.

Initially, after an EFS file system has just been created, inheritance is not yet turned on. You need to activate it explicitly using the `efsmgr` command. After inheritance is turned on, all files that are created inherit this property and are implicitly encrypted. Directories themselves are never encrypted; they only inherit encryption.

You can choose to set inheritance either at the file system level or at directory level. The effects of turning on inheritance are as follows:

► When inheritance is set on a file system, all files that are newly created in this file system are encrypted using the cipher specified at file system creation. All newly created directories inherit the file system cipher.

► When inheritance is set on a directory, all files that are newly created in this directory are encrypted using the cipher inherited from the directory. All newly created subdirectories inherit the parent directory cipher.

If inheritance is set at both file system level and directory level, but is set to use one cipher at file system level and another cipher at the directory level, the files are encrypted using the directory cipher.

After you disable the inheritance for a directory or a file system, all new files that are created are implicitly in clear format. Disabling the inheritance does not affect files that have already been created in encrypted format. They will remain encrypted.

In Example 2-6 on page 69, we show how to turn on inheritance at the file system level and its impact on directories and files as follows:

► The `crfs -v jfs2 -g rootvg -m /efs -a size=100M -a efs=yes` command creates an encrypted file system.

► Trying to set inheritance with the `efsmgr -s -E /efs` command for an unmounted file system fails.

► The `mount /efs` command mounts the file system, and inheritance is successfully set with the `efsmgr -s -E /efs` command.

► The content of the keystore is loaded in the current shell with the `efskeymgr -o ksh` command.

► The `mkdir` command creates the dir1 directory.

► The `efsmgr -E dir1` command sets inheritance for dir1.

► The `getea` and `efsmgr -L` commands show that inheritance has been set.

► file1 is created and the `efskeymgr -l file1` command shows that file1 is encrypted.

► The `efsmgr -D dir1` command disables inheritance for dir1.

► The `efsmgr -L` and `getea` commands show that inheritance has been disabled.

► file2 is created in cleartext format, as shown by the `ls -U` command.

► file1 can be individually decrypted using the `efsmgr -d` command.

► file2 can be individually encrypted using the `efsmgr -e` command. In other words, files can be individually encrypted or decrypted irrespective of the inheritance of their parent directory.

► The new values for file encryption flags are shown by the `ls -U` command.

Example 2-6   Setting and using inheritance at the file system level

```
# crfs -v jfs2 -g rootvg -m /efs -a size=100M -a efs=yes
File system created successfully.
106288 kilobytes total disk space.
New File System size is 212992
# efsmgr -s -E /efs
/efs:
        Error setting EFS attributes: The media is not formatted or the
format is not correct.
# mount /efs
# efsmgr -s -E /efs
# efskeymgr -o ksh
root's EFS password:
#mkdir dir1
# efsmgr -E dir1
# getea dir1
EAName: øSYSTEMø_NRE
EAValue:
# efsmgr -L dir1
EFS inheritance is set with algorithm: AES_128_CBC
# cd dir1
# touch file1
# efsmgr -l file1
EFS File information:
 Algorithm: AES_128_CBC
List of keys that can open the file:
 Key #1:
  Algorithm       : RSA_1024
  Who             : uid 0
  Key fingerprint : 569ae24b:c2da3415:6e7e651e:b7d2f29f:1dda5ab9
# efsmgr -D dir1
# efsmgr -L dir1
Error getting EFS attributes: Cannot find the requested security
attribute.
# getea dir1
dir1 does not have any extended attributes.
# touch file2
# ls -U file2
-rw-r--r---    1 root      system              0 May 14 13:19 file2
# efsmgr -d file1
# efsmgr -e file2
# ls -U file*
-rw-r--r---    1 root      system              0 May 14 13:22 file1
-rw-r--r--e    1 root      system              0 May 14 13:22 file2
```

When you want to enable the encryption inheritance for a file system that has already had inheritance set, you get the following message:

```
# efsmgr -s -E /efs1
/efs1: The EFS attribute is already defined
```

2.4 Encryption at file level

Each file is encrypted using the AES algorithm. AES uses a unique symmetric encryption key that is randomly generated at the time the file is created. The other parameters required by AES, such as mode and key length, are inherited from the directory or the file system in which the file resides.

Data from the file is encrypted when it is written on the disk and decrypted when it is read from the disk.

The symmetric key used to encrypt the file is encrypted with the public key of the user that created the file and stored in file extended attributes. This represents the file cryptographic metadata. If a user or a group is granted access to the file, the file symmetric key is encrypted with the user or group public key and added to the extended attributes of the file. There is virtually no limit for the number of users or groups that can be granted access to the file.

Only a user that has a private key that matches one of the public keys can gain access to the data.

2.4.1 Creating encrypted files and the umask command

When you create an encrypted file, the file symmetric encryption key is encrypted with the public key of the file owner and written in the file's cryptographic metadata. If the `umask` value allows group access, another copy of the file symmetric encryption key is encrypted with the public key of the group and added to the file's cryptographic metadata.

The creation of default file access key and the way it is influenced by the `umask` value is shown in Example 2-7 on page 71. We take the following steps:

► user2 is a member of group1.

► The umask value is 022.

► file1 is created. Since file1 is read-accessible by both user2 and group1, two copies of file symmetric encryption key, one encrypted with the user2 public key and one encrypted with the group1 public key, are added to the file's cryptographic metadata.

- The **umask** value is changed and members of group1 do not have access anymore.
- file2 is created. Since file2 is now accessible only by user2, only one copy of the file symmetric encryption key is encrypted with the user2 public key and added to the file's cryptographic metadata.

Example 2-7   Default file access key and umask

```
$ id
uid=205(user2) gid=203(group1)
$ umask
022
$ touch file1
$ ls -aU file1
-rw-r--r--e    1 user2    group1             0 May 15 14:52 file1
$ efsmgr -l file1
EFS File information:
 Algorithm: AES_128_CBC
List of keys that can open the file:
 Key #1:
  Algorithm       : RSA_1024
  Who             : uid 205
  Key fingerprint : f9d25800:dae0c93b:7695424c:6c617cd6:53b3a7aa
 Key #2:
  Algorithm       : RSA_1024
  Who             : gid 203
  Key fingerprint : 4a73c007:bb0e6b35:abe672c6:cdaf2964:5dd69586
$ umask 077
$ touch file2
$ ls -aU file2
-rw-------e    1 user2    group1             0 May 15 14:53 file2
$ efsmgr -l file2
EFS File information:
 Algorithm: AES_128_CBC
List of keys that can open the file:
 Key #1:
  Algorithm       : RSA_1024
  Who             : uid 205
  Key fingerprint : f9d25800:dae0c93b:7695424c:6c617cd6:53b3a7aa
```

2.4.2 Listing file encryption information

There are several commands that can be used to display file security information, named extended attribute information, and encryption information.

In Example 2-8, we use several commands to find information regarding file encryption as follows:

- ► The `ls -U` command displays file security information. The *e* displayed at the end of file permissions indicates the file is encrypted.

- ► The `getea` command displays the named extended security attributes that shows the file is encrypted. You can see that the EA contains cryptographic metadata for each file.

- ► The `efsmgr -l` command displays the cipher algorithm used to encrypt the file, the algorithm mode, and the length of the key. It also displays the fingerprint of the keys that can access the file along with uid/gid of user/group that own these keys.

Example 2-8   Listing information about file encryption

```
# ls -U file
-rw-r--r--e    1 root      system              0 May 09 19:36 file
# getea file
EAName: øSYSTEMø_NRE
EAValue:
# efsmgr -l file
EFS File information:
 Algorithm: AES_128_CBC
List of keys that can open the file:
 Key #1:
  Algorithm      : RSA_1024
  Who            : uid 0
  Key fingerprint : 5363bd50:0ed7631d:08a93ee5:efdbde20:54f1028c
 Key #2:
  Algorithm      : RSA_1024
  Who            : gid 202
  Key fingerprint : 19f16934:20a54e8e:d59aea33:111a37bf:06261785
```

2.4.3 Implication of encryption on file size and location of disk blocks

In this section, we study the impact of file encryption on file size, inode, and disk blocks. We use a EFS-enabled file system named /efs. We take the following steps, as shown in Example 2-9.

► The **ls -iU file** command shows a small encrypted file named file that is 5 bytes long and corresponds to inode 8.

► The **istat 8 /dev/fslv00** command shows a pointer to block 2f.

► The **du file** command reports that 16 blocks are being used to store this file.

► The **efsmgr -d file** command decrypts the file.

► The **ls -iU file** command shows that file has been decrypted and its size has not changed. The file has been allocated a new inode, inode 5.

► The **istat 5 /dev/fslv00** command now shows a pointer to block 2c.

► The **du file** command now reports that eight blocks are now being used to store this file.

Example 2-9   Effects of encryption on the file

```
# ls -iU file
    8 -rw-r--r--e    1 root     system              5 May 01 01:36 file
# istat 8 /dev/fslv00
Inode 8 on device 10/10 File
Protection: rw-r--r--
Owner: 0(root)          Group: 0(system)
Link count:   1         Length 5 bytes

Last updated:   Tue May  1 01:36:23 MST 2007
Last modified:  Tue May  1 01:36:23 MST 2007
Last accessed:  Tue May  1 01:36:37 MST 2007

Block pointers (hexadecimal):
2f

# du file
16      file
# efsmgr -d file
# ls -iU file
    5 -rw-r--r---    1 root     system              5 May 01 01:41 file
# istat 5 /dev/fslv00
Inode 5 on device 10/10 File
Protection: rw-r--r--
Owner: 0(root)          Group: 0(system)
```

```
Link count:   1          Length 5 bytes

Last updated:   Tue May  1 01:41:38 MST 2007
Last modified:  Tue May  1 01:41:38 MST 2007
Last accessed:  Tue May  1 01:41:38 MST 2007

Block pointers (hexadecimal):
2c

# du file
8       file
#
```

There are some scenarios in which you should take this behavior into account.

For example, if you are running a commercial Web server that has a large number of small files in cache and the data in these files needs to be protected by encryption, you need to closely monitor the storage space available.

Another scenario is an environment where a lot of file encryption/decryption operations take place and the corresponding inode number of each file has a particular importance for other applications. Pay special attention if you back up your files using an inode number.

Since the processes of encryption and decryption change the inode number that is allocated to a file, the time of last inode update, file modification, and file reference is modified as well.

2.4.4 Looking at disk blocks of an encrypted file

In this section, we study the impact of file encryption on corresponding disk blocks. We use a EFS-enabled file system named /efs. We take the following steps, as shown in Example 2-10 on page 75.

► We create a cleartext file with a easy to recognize content within the /efs encrypted file system. The inode corresponding to the file is 19.

► We look the information contained in the inode 19 and find that the file uses block number 3d.

► We use **fsdb** to look at the content of block 3d and notice that data is stored on the disk in clear format.

► We encrypt the file using the **efsmgr** command and observe that a new inode, inode, 4 has been associated to the file.

- ▶ We look at the information contained in the inode 19 and find that the file uses block number 2d.
- ▶ We use **fsdb** to look at the content of block 2d and find that data is stored on the disk in encrypted format.

Example 2-10   Examining disk blocks of an encrypted file

```
root@nimrod:/efs# ls -ailU
total 24
    2 drwxrwxrwxe    3 root     system        256 May 08 16:49 ./
    2 drwxr-xr-x-   24 root     system       4096 May 08 14:37 ../
   19 -rw-r--r---    1 root     system        144 May 08 16:36 file
    3 drwxr-xr-x-    2 root     system        256 May 07 11:27
lost+found/
root@nimrod:/efs# cat file
111111111111111
222222222222222
333333333333333
444444444444444
555555555555555
666666666666666
777777777777777
888888888888888
999999999999999
root@nimrod:/efs# istat 19 /dev/fslv01
Inode 19 on device 10/12       File
Protection: rw-r--r--
Owner: 0(root)         Group: 0(system)
Link count:   1        Length 144 bytes

Last updated:   Tue May  8 16:36:34 CDT 2007
Last modified:  Tue May  8 16:36:34 CDT 2007
Last accessed:  Tue May  8 16:51:44 CDT 2007

Block pointers (hexadecimal):
3d
root@nimrod:/efs# fsdb /dev/fslv01
Filesystem /dev/fslv01 is mounted.  Modification is not permitted.

File System:                   /dev/fslv01

File System Size:              212576  (512 byte blocks)
Aggregate Block Size:          4096
Allocation Group Size:         8192    (aggregate blocks)
```

```
> display 0x3d
Block: 61      Real Address 0x3d000
00000000:  31313131 31313131 31313131 3131310A  |111111111111111.|
00000010:  32323232 32323232 32323232 3232320A  |222222222222222.|
00000020:  33333333 33333333 33333333 3333330A  |333333333333333.|
00000030:  34343434 34343434 34343434 3434340A  |444444444444444.|
00000040:  35353535 35353535 35353535 3535350A  |555555555555555.|
00000050:  36363636 36363636 36363636 3636360A  |666666666666666.|
00000060:  37373737 37373737 37373737 3737370A  |777777777777777.|
00000070:  38383838 38383838 38383838 3838380A  |888888888888888.|
00000080:  39393939 39393939 39393939 3939390A  |999999999999999.|
00000090:  00000000 00000000 00000000 00000000  |...............|
000000a0:  00000000 00000000 00000000 00000000  |...............|
000000b0:  00000000 00000000 00000000 00000000  |...............|
000000c0:  00000000 00000000 00000000 00000000  |...............|
000000d0:  00000000 00000000 00000000 00000000  |...............|
000000e0:  00000000 00000000 00000000 00000000  |...............|
000000f0:  00000000 00000000 00000000 00000000  |...............|
-hit enter for more-
... lines skipped.......................................................
root@nimrod:/efs# efsmgr -e file
root@nimrod:/efs# ls -ialU
total 32
    2 drwxrwxrwxe    3 root     system          256 May 08 17:02 ./
    2 drwxr-xr-x-   24 root     system         4096 May 08 14:37 ../
    4 -rw-r--r--e    1 root     system          144 May 08 17:02 file
    3 drwxr-xr-x-    2 root     system          256 May 07 11:27
lost+found/
root@nimrod:/efs# istat 4 /dev/fslv01
Inode 4 on device 10/12 File
Protection: rw-r--r--
Owner: 0(root)          Group: 0(system)
Link count:   1         Length 144 bytes

Last updated:   Tue May  8 17:02:49 CDT 2007
Last modified:  Tue May  8 17:02:49 CDT 2007
Last accessed:  Tue May  8 17:02:49 CDT 2007

Block pointers (hexadecimal):
2d
root@nimrod:/efs# fsdb /dev/fslv01
Filesystem /dev/fslv01 is mounted.  Modification is not permitted.

File System:                    /dev/fslv01
```

```
File System Size:          212576  (512 byte blocks)
Aggregate Block Size:      4096
Allocation Group Size:     8192    (aggregate blocks)

> display 0x2d
Block: 45      Real Address 0x2d000
00000000:  484AC9D2 C83AE1AB EB598765 A1434DCC   |HJ...:...Y.e.CM.|
00000010:  631ABD5A F41D5081 C88B154F 4776DD7B   |c..Z..P....OGv.{|
00000020:  9ED051A5 9E4AD8C4 A1405607 2681ECE7   |..Q..J...@V.&...|
00000030:  F736D795 32D183CF 1C30CD86 EB9009A2   |.6..2....0......|
00000040:  7A3D3142 3BFC6196 86490E10 4203FD52   |z=1B;.a..I..B..R|
00000050:  57F4E7A7 66C5A9CD E9E5CD71 8B64C368   |W...f......q.d.h|
00000060:  391963FA F2ED5A07 0A1BDEEF 20E9E8A4   |9.c...Z..... ...|
00000070:  85F888EF 32E95FCF 4BF03826 D62393C1   |....2._.K.8&.#..|
00000080:  DD138A6E 4E70FD23 EF691605 ACCCDB01   |...nNp.#.i......|
00000090:  AD59A23A F0D9E3E6 F32575EF 5DD8EF15   |.Y.:.....%u.]...|
000000a0:  0DA37573 0DFC54CE 505983F0 86648058   |..us..T.PY...d.X|
000000b0:  DDF4EB20 F6A3C1DD 1A1FC8BD CFDFA163   |... ...........c|
000000c0:  989EE442 5A82B98A 2AE34A77 0FE41338   |...BZ...*.Jw...8|
000000d0:  5E7C14DC D5C56B9E 093CC234 B9AF879E   |^|....k..<.4....|
000000e0:  19638EBB A13C4154 5AE06F68 62B62326   |.c...<ATZ.ohb.#&|
000000f0:  91F6C0E5 5F9D87A8 D50DAA59 9662B6DE   |...._......Y.b..|
-hit enter for more-
```

2.4.5 Decrypting a file

In this section, we show how to decrypt an encrypted file. We use a EFS-enabled
file system named /efs. We take the following steps as shown in Example 2-11 on
page 78:

▶ The `efsmgr -l` command displays encryption information about file.

▶ Decrypt the file using the `efsmgr -d` command.

▶ The `ls` command shows the file is no longer encrypted.

▶ The `getea` command shows the file does not have any extended attributes,
   and therefore no cryptographic metadata.

▶ The `efsmgr -l` command displays an error since it cannot get a security
   attribute.

Example 2-11   Decrypting an encrypted file

```
# efsmgr -l file
EFS File information:
 Algorithm: AES_128_CBC
List of keys that can open the file:
 Key #1:
  Algorithm       : RSA_1024
  Who             : uid 0
  Key fingerprint : 5131afb9:95232d08:ae5c499d:dd9941fe:90163a89
# efsmgr -d file
# ls -U file
-rw-r--r---    1 root     system          21 Apr 24 06:18 file
# getea file
file does not have any extended attributes.
# efsmgr -l file
Error getting EFS attributes: Cannot find the requested security
attribute.
```

Even if you are the owner of a file, you must have the private key to be able to
decrypt the file, as shown in Example 2-12. An attempt to decrypt the file without
having the access key fails.

Example 2-12   Private key must be loaded into the process in order to decrypt a file

```
# id
uid=0(root) gid=0(system)
groups=2(bin),3(sys),7(security),8(cron),10(audit),11(lp)
# ls -U file
-rw-r--r--e    1 root     system          21 Apr 24 06:29 file
# efskeymgr -V
There is no key loaded in the current process.
# cat file
cat: 0652-050 Cannot open file.
# efskeymgr -o ksh
root's EFS password:
# efskeymgr -d file
# ls -U file
-rw-r--r--- 1 root      system          21 Apr 24 06:29 file
```

2.4.6 Encrypting a file

In this section, we show how to encrypt an encrypted file. We use an EFS-enabled file system named /efs. We take the following steps shown in Example 2-13:

► The **ls -U** command shows the file is no longer encrypted.

► The **efsmgr -e** command encrypts the file.

► The **efsmgr -l** command displays encryption information about the file.

Example 2-13   Encrypting a file

```
# ls -U file
-rw-r--r---    1 root     system           21 Apr 24 06:18 file
# efsmgr -e file
# efsmgr -l file
EFS File information:
 Algorithm: AES_128_CBC
List of keys that can open the file:
 Key #1:
  Algorithm      : RSA_1024
  Who            : uid 0
  Key fingerprint : 5131afb9:95232d08:ae5c499d:dd9941fe:90163a89
```

Even if you are the owner of a file, you must have the private key to be able to encrypt the file, as shown in Example 2-14. An attempt to encrypt the file without having the access key fails.

Example 2-14   Private key must be loaded into the process in order to encrypt a file

```
# efskeymgr -V
There is no key loaded in the current process.
# ls -U ffff
-rw-r--r---    1 root     system              0 Apr 25 00:09 ffff
# efsmgr -e ffff
./.efs.9Xqi7a: Cannot find the requested security attribute.
# efskeymgr -o ksh
root's EFS password:
#efsmgr -e ffff
# efsmgr -l ffff
EFS File information:
 Algorithm: AES_256_CBC
List of keys that can open the file:
 Key #1:
  Algorithm       : RSA_1024
  Who             : uid 0
  Key fingerprint : 5131afb9:95232d08:ae5c499d:dd9941fe:90163a89
```

2.4.7 Changing file encryption key parameters

In Example 2-15 on page 81, we show how to change the length of encryption key and the AES mode for an encrypted file. We use a EFS-enabled file system named /efs. The initial key length is 128 and the initial mode is CBC. A new key having the length of 256 is generated and the file is encrypted in ECB mode.

Example 2-15   Changing the algorithm of file encryption

```
# efsmgr -l file
EFS File information:
 Algorithm: AES_128_CBC
List of keys that can open the file:
 Key #1:
  Algorithm       : RSA_1024
  Who             : uid 0
  Key fingerprint : 5363bd50:0ed7631d:08a93ee5:efdbde20:54f1028c
# efsmgr -t file -c AES_256_ECB
# efsmgr -l file
EFS File information:
 Algorithm: AES_256_ECB
List of keys that can open the file:
 Key #1:
  Algorithm       : RSA_1024
  Who             : uid 0
  Key fingerprint : 5363bd50:0ed7631d:08a93ee5:efdbde20:54f1028c
```

Attention: If you change the key parameters or just the key, the file is assigned a different inode. See 2.4.3, "Implication of encryption on file size and location of disk blocks" on page 73 for more details.

2.4.8 File access permissions

It is very important to understand how traditional AIX file permissions interact with file encryption in terms of controlling file access. These two mechanisms do not overlap. They cooperate to offer a even more granular control for file access.

When trying to use an encrypted file, Discretionary Access Control (DAC) and Access Control List (ACL) are checked first for the file access permission. Only if the access is granted at these levels does the key verification proceed.

Example 2-16 on page 82 shows how traditional file access permissions are checked first. We took the following steps in our scenario:

► Log in as user1. user1 has uid 210.

► The `ls` command shows that the file is owned by user1. There is no read or write access for users in the group staff.

► The `efsmgr -l` command displays the signatures for the keys that can access the file. One key belongs to a user with uid 210 (user1) and the second belongs to a user with uid 211 (user2)

- ▶ user2 logs in and the `efsmgr -V` command proves he or she has the access key.

- ▶ Although user2 has the key, the `cat` command fails because user2 cannot read the file (he or she has no read access).

- ▶ user1 logs in and grants read access to all users from group staff using the `chmod` command.

- ▶ The `cat` command completes successfully because user2 has now both read access and the key.

- ▶ user2 tries to append some text to the file to the file using the `echo` command and fails, as he or she does not have write access.

- ▶ user1 logs in and grants write access to all users from the group staff using the `chmod` command.

- ▶ user2 can now successfully append text to the file because he or she now has both write access and the key.

Example 2-16   File access permissions and file security do not overlap

```
$ id
uid=210(user1) gid=1(staff)
$ ls -U file
-rwx------e    1 user1    staff              38 Apr 26 07:44 file
$ efsmgr -l file
EFS File information:
 Algorithm: AES_256_CBC
List of keys that can open the file:
 Key #1:
  Algorithm      : RSA_1024
  Who            : uid 210
  Key fingerprint : 3cf44e44:2544568e:9377638d:adce70ca:5fd1c54d
 Key #2:
  Algorithm      : RSA_1024
  Who            : uid 211
  Key fingerprint : a9e9087f:82a41b75:a258f918:8de0c180:38028ac2
$ id
uid=211(user2) gid=1(staff)
$ efskeymgr -V
List of keys loaded in the current process:
 Key #0:
    Kind ................... User key
    Id   (uid / gid) ......... 211
    Type ................... Private key
    Algorithm ............... RSA_1024
    Validity ............... Key is valid
```

```
    Fingerprint ..............
a9e9087f:82a41b75:a258f918:8de0c180:38028ac2
$ cat /home/user1/file
cat: 0652-050 Cannot open /home/user1/file.
$ id
uid=210(user1) gid=1(staff)
$ chmod g+r file
$ id
uid=211(user2) gid=1(staff)
$ cat /home/user1/file
If you can see this message it is ok
$ echo "Another message in the file " >> /home/user1/file
The file access permissions do not allow the specified action.
ksh: /home/user1/file: 0403-005 Cannot create the specified file.
$ id
uid=210(user1) gid=1(staff)
$ chmod g+w file
$ id
uid=211(user2) gid=1(staff)

$ echo "Another message in the file " >> /home/user1/file
$ cat /home/user1/file
If you can see this message it is ok

Another message in the file
$
```

When you use the **chmod** command to grant group permissions for a file, if the
group public access key is already contained in the file cryptographic metadata, it
will be automatically added. When you remove all group file permissions, the
public group key is also removed from file cryptographic metadata, as shown in
Example 2-17 on page 84:

► There are no group permissions for the file.

► Only user2 has the access key for the file shown by the **efsmgr -l** command.

► Write access is granted to members of group1 using the **chmod** command.

► The **efsmgr -l** command shows that another copy of the file symmetric
  encryption key is encrypted with the public key of group1 and added to file
  cryptographic metadata.

► Write access has been removed from members of group1 using the **chmod**
  command.

► The **efsmgr -l** command shows that the copy of the file symmetric encryption key that has been encrypted with the public key of the group was removed from file cryptographic metadata.

Example 2-17   Group access permissions and group access key

```
$ ls -U file
-rw-------e   1 user2     group1              0 May 15 15:32 file
$ efsmgr -l file
EFS File information:
 Algorithm: AES_128_CBC
List of keys that can open the file:
 Key #1:
  Algorithm      : RSA_1024
  Who            : uid 205
  Key fingerprint : f9d25800:dae0c93b:7695424c:6c617cd6:53b3a7aa
$ chmod g+w file
$ efsmgr -l file
EFS File information:
 Algorithm: AES_128_CBC
List of keys that can open the file:
 Key #1:
  Algorithm      : RSA_1024
  Who            : uid 205
  Key fingerprint : f9d25800:dae0c93b:7695424c:6c617cd6:53b3a7aa
 Key #2:
  Algorithm      : RSA_1024
  Who            : gid 203
  Key fingerprint : 4a73c007:bb0e6b35:abe672c6:cdaf2964:5dd69586
$ chmod g-w file
$ ls -U file
-rw-------e   1 user2     group1              0 May 15 15:32 file
$ efsmgr -l file
EFS File information:
 Algorithm: AES_128_CBC
List of keys that can open the file:
 Key #1:
  Algorithm      : RSA_1024
  Who            : uid 205
  Key fingerprint : f9d25800:dae0c93b:7695424c:6c617cd6:53b3a7aa
```

2.4.9 Changing file ownership

In this section, we study the interaction between file ownership and cryptographic file metadata. We use an EFS-enabled file system named /efs, two users named user2 and user3 that have access to the /efs file system, and a test file named file that is initially owned by user2. We run the scenario in root admin mode and do the following steps that are also shown in Example 2-18:

► Log in as user2. user2 has uid 204.

► The `ls` command shows that the file is encrypted and is owned by user2.

► The `efsmgr -l` file displays the signature for the key that can access file. The key belongs to the user with uid 204 (user2).

► The `efskeymgr -V` command shows that user2 has the access key and the `cat` command can be successfully used to read the file.

► Root logs in and changes file ownership using the `chown` command.

► The `ls` command shows that the file ownership has been changed.

► The `efsmgr -l` command displays the signature for the new key that can access the file. The key belongs to the user with uid 205. The key belonging to user2 has been removed from the file cryptographic metadata.

► user3 logs in. He or she has uid 205.

► The `efsmgr -V` command proves that user3 has the access key for file.

► The `cat` command can be successfully used to read the file.

► user2 does not have access to the file anymore and an attempt to read the file using the `cat` command fails.

Example 2-18   Changing the ownership of a file

```
$ id
uid=204(user2) gid=1(staff)
$ ls -U file
-rw-r--r--e   1 user2    staff             23 May 10 10:13 file
$ efsmgr -l file
EFS File information:
 Algorithm: AES_128_CBC
List of keys that can open the file:
 Key #1:
  Algorithm      : RSA_1024
  Who            : uid 204
  Key fingerprint : 366da13e:e40d8c52:0d0492ce:3b8797b6:6f5f9bf2
$ efskeymgr -V
List of keys loaded in the current process:
 Key #0:
```

```
   Kind .................... User key
   Id   (uid / gid) ........ 204
   Type .................... Private key
   Algorithm ............... RSA_1024
   Validity ................ Key is valid
   Fingerprint .............
366da13e:e40d8c52:0d0492ce:3b8797b6:6f5f9bf2
$ cat file
data in encrypted file
# id
uid=0(root) gid=0(system)
groups=2(bin),3(sys),7(security),8(cron),10(audit),11(
lp)
# chown user3:staff file
# ls -U file
-rw-r--r--e   1 user3    staff            23 May 10 10:13 file
# efsmgr -l file
EFS File information:
 Algorithm: AES_128_CBC
List of keys that can open the file:
 Key #1:
  Algorithm       : RSA_1024
  Who             : uid 205
  Key fingerprint : 30412121:e5a7b90d:dba7dd19:2c45b1e0:c331c09f
$ id
uid=205(user3) gid=1(staff)
$ efskeymgr -V
List of keys loaded in the current process:
 Key #0:
   Kind .................... User key
   Id   (uid / gid) ........ 205
   Type .................... Private key
   Algorithm ............... RSA_1024
   Validity ................ Key is valid
   Fingerprint .............
30412121:e5a7b90d:dba7dd19:2c45b1e0:c331c09f
$ cat file
data in encrypted file
$ id
uid=204(user2) gid=1(staff)
$ cat file
cat: 0652-050 Cannot open file.
```

If you also change the group when you change file ownership, the symmetric file
key encrypted with the new group public key is added to file cryptographic
metadata, as shown in Example 2-19.

Example 2-19   Changing file ownership

```
# ls -U file
-rw-r--r--e     1 user1     group1             5 May 13 15:03 file
# efsmgr -l file
EFS File information:
 Algorithm: AES_128_CBC
List of keys that can open the file:
 Key #1:
  Algorithm       : RSA_1024
  Who             : uid 210
  Key fingerprint : ea2d177c:7776d3a2:c840b44b:1c2b1ccf:c875d55d
 Key #2:
  Algorithm       : RSA_1024
  Who             : gid 204
  Key fingerprint : 53cd3824:7c9d508e:b825a253:f3209fcf:76f1766f
# chown user2:group2 file
# ls -U file
-rw-r--r--e     1 user2     group2             5 May 13 15:03 file
# efsmgr -l file
EFS File information:
 Algorithm: AES_128_CBC
List of keys that can open the file:
 Key #1:
  Algorithm       : RSA_1024
  Who             : uid 211
  Key fingerprint : 52dccf67:22ab2bc9:4caa4ec8:7ceb8675:972c1404
 Key #2:
  Algorithm       : RSA_1024
  Who             : gid 205
  Key fingerprint : 61e6538d:343c8313:df807f20:ab02451e:734309ca
```

2.4.10 Granting a user or a group access to a file

In this section, we explain and show how users and group members can be
granted access to files.

When a user/group is granted access to a file, a copy of the symmetric key used
to encrypt the file is encrypted with the public key of the user/group and added to
the file cryptographic metadata.

We use the following scenario: an EFS-enabled file system named /efs, three users named user1, user2, and user3 that have access to the /efs file system, and a test file named file, which is owned by user2. user1 is member of group1. We run our scenario in root admin mode and take the following steps shown in Example 2-20:

► Log in as user2. user2 has uid 204.

► The `ls` command shows that the file is encrypted and is owned by user2. It is very important to understand that in root admin mode, only root and the owner of the file can grant other users access.

► The `efsmgr -l` command displays the signature for the keys that can access the file. There is only one key that belongs to the user with uid 204 (user2).

► The `efskeymgr -V` command shows user2 has the access key that is required for granting access to the file.

► user2 grants user3 access to the file using the `efsmgr` command.

► The `efsmgr -l` command displays the signature for the key that can access the file. The new key that has been added belongs to the user with uid 205 (user3).

► user2 grants group1 access to the file using the `efsmgr` command.

► The `efsmgr -l` command displays the signature for the keys that can access the file. The new key that has been added belongs to group with gid 202 (group1). File cryptographic metadata has been updated.

► user3 logs in. He or she has uid 203.

► The `efsmgr -V` command shows user3 has a user access key that matches one of the keys that are granted access to the file.

► The `cat` command can be successfully used to read the file.

► user1 logs in. He or she has uid 205.

► the `efsmgr -V` command shows that user1 has a group (group1) access key that matches one of the keys that are granted access to the file.

► The `cat` command can be successfully used to read the file.

► user1 maliciously tries to grant user5 access to the file using the `efsmgr` command and fails. In other words, a user that has been granted access to file by the file owner cannot further grant access for a third party.

Example 2-20   Granting a user and a group access to a file

```
$ id
uid=204(user2) gid=1(staff)
$ ls -U file
-rw-r--r--e    1 user2    staff           23 May 10 10:55 file
```

```
$ efsmgr -l file
EFS File information:
 Algorithm: AES_128_CBC
List of keys that can open the file:
 Key #1:
  Algorithm      : RSA_1024
  Who            : uid 204
  Key fingerprint : 366da13e:e40d8c52:0d0492ce:3b8797b6:6f5f9bf2
$ efskeymgr -V
List of keys loaded in the current process:
 Key #0:
   Kind .................... User key
   Id   (uid / gid) ........ 204
   Type ................... Private key
   Algorithm .............. RSA_1024
   Validity ............... Key is valid
   Fingerprint ............
366da13e:e40d8c52:0d0492ce:3b8797b6:6f5f9bf2
$ efsmgr -a file -u user3
$ efsmgr -l file
EFS File information:
 Algorithm: AES_128_CBC
List of keys that can open the file:
 Key #1:
  Algorithm      : RSA_1024
  Who            : uid 204
  Key fingerprint : 366da13e:e40d8c52:0d0492ce:3b8797b6:6f5f9bf2
 Key #2:
  Algorithm      : RSA_1024
  Who            : uid 205
  Key fingerprint : 30412121:e5a7b90d:dba7dd19:2c45b1e0:c331c09f
$ efsmgr -a file -g group1
$ efsmgr -l file
EFS File information:
 Algorithm: AES_128_CBC
List of keys that can open the file:
 Key #1:
  Algorithm      : RSA_1024
  Who            : uid 204
  Key fingerprint : 366da13e:e40d8c52:0d0492ce:3b8797b6:6f5f9bf2
 Key #2:
  Algorithm      : RSA_1024
  Who            : uid 205
  Key fingerprint : 30412121:e5a7b90d:dba7dd19:2c45b1e0:c331c09f
 Key #3:
```

```
        Algorithm      : RSA_1024
        Who            : gid 202
        Key fingerprint : 19f16934:20a54e8e:d59aea33:111a37bf:06261785
$ id
uid=205(user3) gid=1(staff)
$ efskeymgr -V
List of keys loaded in the current process:
 Key #0:
    Kind .................... User key
    Id   (uid / gid) ........ 205
    Type .................... Private key
    Algorithm ............... RSA_1024
    Validity ................ Key is valid
    Fingerprint .............
30412121:e5a7b90d:dba7dd19:2c45b1e0:c331c09f
$ cat file
data in encrypted file
$ id
uid=203(user1) gid=1(staff) groups=202(group1)
$ efskeymgr -V
List of keys loaded in the current process:
 Key #0:
    Kind .................... User key
    Id   (uid / gid) ........ 203
    Type .................... Private key
    Algorithm ............... RSA_1024
    Validity ................ Key is valid
    Fingerprint .............
eb1aab3e:39c3191c:15cb36d6:57bb2a7c:b3c6d356
 Key #1:
    Kind .................... Group key
    Id   (uid / gid) ........ 202
    Type .................... Private key
    Algorithm ............... RSA_1024
    Validity ................ Key is valid
    Fingerprint .............
19f16934:20a54e8e:d59aea33:111a37bf:06261785
$ cat file
data in encrypted file
$ efsmgr -a file -u user5
file: The file access permissions do not allow the specified action.
Cannot open or lock the file.
```

> **Attention:** Trying to give access to a user/group that is defined on the system but does not have a keystore created yet fails because their is no public key to encrypt the file symmetric encryption key and add to file cryptographic metadata. Any attempt displays an error message similar to the following:
>
> ```
> $efsmgr -a file -u user5
> Unable to get public key from user "user5" (skipped): (Keystore does
> not exist)
> user5: The system call does not exist on this system.
> ```

2.4.11 Revoking a user or group access to a file

In this section, we explain and show how user access to files can be revoked.

When a user's/group's access to a file is revoked, the existing copy of the symmetric key used to encrypt the file that has been encrypted with the public key of the user/group is removed from the file cryptographic metadata.

We use the following scenario: an EFS-enabled file system named /efs, three users named user1, user2, and user3 that have access to the /efs file system, and a test file named file, which is owned by user2. user1 is member of group1. user1 and group1 have been granted access to the file. We run our scenario in root admin mode and take the following steps shown in Example 2-21 on page 92:

► Log in as user2. user2 has uid 204.

► The `ls` command shows that the file is encrypted and is owned by user2.

► The `efsmgr -l` command displays the signature for the keys that can access the file. There are three keys that can access the file: one key belongs to the user with uid 204 (user2), one key belongs to the user with uid 205 (user3), and one key belongs to the group with gid 202 (group1).

► user3 logs in and tries to revoke group1 access to the file using the `efsmgr` command and fails. In other words, a user that has access to the file granted by the file owner cannot revoke access for a third party.

► The `efskeymgr -V` command shows that user2 has the access key that is required for revoking access to the file.

► user2 revokes user3 access to the file using the `efsmgr` command.

► The `efsmgr -l` command displays the signature for the keys that can still access file. File cryptographic metadata has been updated and the key that belongs to user with uid 205 (user3) has been deleted.

► user3 logs in, but can no longer access the file and the `cat` command fails.

- ► user2 revokes group1 access to the file using the **efsmgr** command.
- ► The **efsmgr -l** command displays the signature for the keys that can still access the file. File cryptographic metadata has been updated and the key that belongs to user with gid 202 (group1) has been deleted.
- ► user1 can no longer access the file and the **cat** commands fails.
- ► user2 tries and fails to revoke his or her own access. In other words, you cannot revoke your own access.

Example 2-21   Revoking user and group access to a file

```
$ id
uid=204(user2) gid=1(staff)
$ ls -U file
-rw-r--r--e    1 user2    staff            23 May 10 10:55 file
$ efsmgr -l file
EFS File information:
 Algorithm: AES_128_CBC
List of keys that can open the file:
 Key #1:
  Algorithm       : RSA_1024
  Who             : uid 204
  Key fingerprint : 366da13e:e40d8c52:0d0492ce:3b8797b6:6f5f9bf2
 Key #2:
  Algorithm       : RSA_1024
  Who             : uid 205
  Key fingerprint : 30412121:e5a7b90d:dba7dd19:2c45b1e0:c331c09f
 Key #3:
  Algorithm       : RSA_1024
  Who             : gid 202
  Key fingerprint : 19f16934:20a54e8e:d59aea33:111a37bf:06261785
$ id
uid=205(user3) gid=1(staff)
$ efsmgr -r file -g group1
file: The file access permissions do not allow the specified action.
Cannot open or lock the file.
$ id
uid=204(user2) gid=1(staff)
$ efskeymgr -V
List of keys loaded in the current process:
 Key #0:
    Kind .................... User key
    Id   (uid / gid) ........ 204
    Type .................... Private key
    Algorithm ............... RSA_1024
```

```
   Validity ................ Key is valid
   Fingerprint ..............
366da13e:e40d8c52:0d0492ce:3b8797b6:6f5f9bf2
$ efsmgr -r file -u user3
$ efsmgr -l file
EFS File information:
 Algorithm: AES_128_CBC
List of keys that can open the file:
 Key #1:
  Algorithm      : RSA_1024
  Who            : uid 204
  Key fingerprint : 366da13e:e40d8c52:0d0492ce:3b8797b6:6f5f9bf2
Key #2:
  Algorithm      : RSA_1024
  Who            : gid 202
  Key fingerprint : 19f16934:20a54e8e:d59aea33:111a37bf:06261785
$ id
uid=205(user3) gid=1(staff)
$ cat file
cat: 0652-050 Cannot open file.
$ efsmgr -r file -g group1
$ efsmgr -l file
EFS File information:
 Algorithm: AES_128_CBC
List of keys that can open the file:
 Key #1:
  Algorithm      : RSA_1024
  Who            : uid 204
  Key fingerprint : 366da13e:e40d8c52:0d0492ce:3b8797b6:6f5f9bf2
$ id
uid=203(user1) gid=1(staff) groups=202(group1)
$ cat file
cat: 0652-050 Cannot open file.
$ efsmgr -r file -u user2
Error editing file EFS access list: A system call received a parameter
that is not valid.
```

Attention: Trying to revoke access for a user/group that is defined on the system but does not have a keystore is successful because during this operation, the file's symmetric encryption that is encrypted with user/group public key is removed from the file's cryptographic metadata.

2.4.12 Granting/revoking access in root admin mode

The following is illustrated in Example 2-22:

► The `ls` command shows that the file is encrypted and is owned by user2.

► The `efsmgr -l` command displays the signature for the keys that can access the file. There are three keys that can access the file: one key belongs to the user with uid 204 (user2), one key belongs to the user with uid 205 (user3), and one key belongs to a group with gid 202 (group1).

► user2 grants root access to the file.

► root logs in and root revokes access from user3, group1 and user2 to the file despite the fact that the file is owned by user2. In other words, root still has complete control to the file access when running in root admin mode.

► `efsmgr -l` displays the signature for the keys that can still access the file. The file cryptographic metadata has been updated and the only key that can access the file belongs to the user with uid 0 (root).

► Although user2 is the file owner, user2 can no longer access the file and the `cat` command fails.

► user2 cannot grant user3 access to the file.

► user2 cannot revoke root access to the file.

Example 2-22   Granting/removing access in root admin mode

```
$ ls -U file
-rw-r--r--e    1 user2    staff            9 May 10 15:01 file
$ efsmgr -l file
EFS File information:
 Algorithm: AES_128_CBC
List of keys that can open the file:
 Key #1:
  Algorithm      : RSA_1024
  Who            : uid 204
  Key fingerprint : 366da13e:e40d8c52:0d0492ce:3b8797b6:6f5f9bf2
 Key #2:
  Algorithm      : RSA_1024
  Who            : uid 205
  Key fingerprint : 30412121:e5a7b90d:dba7dd19:2c45b1e0:c331c09f
 Key #3:
  Algorithm      : RSA_1024
  Who            : gid 202
  Key fingerprint : 19f16934:20a54e8e:d59aea33:111a37bf:06261785
$ efsmgr -a file -u root
# id
```

```
uid=0(root) gid=0(system)
groups=2(bin),3(sys),7(security),8(cron),10(audit),11(lp)
# efsmgr -r file -u user3
# efsmgr -r file -g group1
# efsmgr -r file -u user2
# efsmgr -l file
EFS File information:
 Algorithm: AES_128_CBC
List of keys that can open the file:
 Key #1:
  Algorithm      : RSA_1024
  Who            : uid 0
  Key fingerprint : 5363bd50:0ed7631d:08a93ee5:efdbde20:54f1028c
$ id
uid=204(user2) gid=1(staff)
$ cat file
cat: 0652-050 Cannot open file.
$ efsmgr -a file -u user3
file: Security authentication is denied.
Cannot open or lock the file.
$ efsmgr -r file -u root
file: Security authentication is denied.
Cannot open or lock the file.
```

2.5 Users management

In this section, we show how user management functions incorporate new cryptographic features.

2.5.1 Defining users

Every time you create a new user, you must define his or her security related information. User security attributes that have been added to /etc/security/user are shown in Example 2-23.

Example 2-23   New user attributes added in user security configuration file

```
default:
        admin = false
        login = true
        su = true
        daemon = true
```

```
                 rlogin = true
                 sugroups = ALL
                 admgroups =
                 ttys = ALL
                 auth1 = SYSTEM
                 auth2 = NONE
                 tpath = nosak
                 umask = 022
                 expires = 0
                 SYSTEM = "compat"
                 logintimes =
                 pwdwarntime = 0
                 account_locked = false
                 loginretries = 0
                 histexpire = 0
                 histsize = 0
                 minage = 0
                 maxage = 0
                 maxexpired = -1
                 minalpha = 0
                 minother = 0
                 minlen = 0
                 mindiff = 0
                 maxrepeats = 8
                 dictionlist =
                 pwdchecks =
                 default_roles =
```

efs_keystore_access = file
efs_adminks_access = file
efs_initialks_mode = admin
efs_allowksmodechangebyuser = yes
efs_keystore_algo = RSA_1024
efs_file_algo = AES_128_CBC

In order to support the new features that have been added for user security, six fields have been also added to the SMIT screens used for user management, as shown in Example 2-24 on page 97.

Example 2-24   New fields added in SMIT panel for user creation

```
Add a User

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

[MORE...52]                                         [Entry Fields]
  Hard NPROC per user                         []
#
  File creation UMASK                              [022]
  AUDIT classes                               []
+
  TRUSTED PATH?                                     nosak
+
  PRIMARY authentication method                    [SYSTEM]
  SECONDARY authentication method                  [NONE]
  Projects                                    []
+
  Keystore Access                                  [file]
+
  Adminkeystore Access                             [file]
+
  Initial Keystore Mode                       [admin]
+
  Allow user to change Keystore Mode?           [yes]
+
  Keystore Encryption Algorithm                    [RSA_1024]
+
  File Encryption Algorithm                        [AES_128_CBC]
+
[BOTTOM]

F1=Help             F2=Refresh          F3=Cancel              F4=List
Esc+5=Reset         Esc+6=Command       Esc+7=Edit             Esc+8=Image
Esc+9=Shell         Esc+0=Exit          Enter=Do
```

The meaning of the newly-added fields in Example 2-24 are:

Keystore Access   The value of this field must be `file` or `none`. File means
                        local /var/efs/user/*. `None` means no keystore.

Adminkeystore Access

                        The field describes the location of the admin keystore.
                        The value of this field must be `file`.

Initial Keystore Mode This field describes the initial mode of the user keystore. The value of this field can be either `admin` or `guard`, corresponding to either root admin mode or root guard mode.

Allow user to change Keystore Mode
This field describes whether the user can change the mode of his or her keystore. The value of this field can be either `yes` or `no`.

Keystore Encryption Algorithm
This field describes the algorithm and the length of the keys used for user private and public keys. If you change the value of this field, the corresponding attribute from the user stanza from /etc/security/user is changed accordingly. The value can be one of the following:

RSA_1024

RSA_2048

RSA_4096

File Encryption Algorithm
This field describes the algorithm, the length of the key, and the mode of the algorithm. If you change the value of this field, the corresponding attribute from the user stanza from /etc/security/user is changed accordingly The value of this field can be one of the following:

AES_128_CBC

AES_128_ECB

AES_192_CBC

AES_192_ECB

AES_256_CBC

AES_256_ECB

When the user is defined by root, the security attributes from user stanza in /etc/security/user are initialized using the default values. At this time, there is no keystore for the user. Only after a password is assigned to the user is his or her keystore created.

Tip: In large environments with a large number of users, it may be difficult for an administrator to manually define a password for each and every user. If you want to define a large number of users, assign them a password and create corresponding user keystores (you can create a script that uses the `chpasswd` command). This command will set a temporary password for the user and the user must change his or her password the first time he or she logs in.

Important: When creating users in root guard mode, a security administrator must understand that roots needs still needs to be considered trustworthy at the time of keystore creation. Also, only root can specify the user keystore to be root admin more or root guard mode and if user is allowed to change the keystore mode.

2.5.2 User keystore

Keystores are containers for public and private data that is used for file encryption/ decryption, for controlling access to files, and for user and group management. For each user a keystore is automatically created when the user is assigned a password. A public/ private key pair is automatically generated and stored in the keystore. Parameters used to generate the public and private key, such as key length, are read from /etc/security/user.

The user keystore may contain, apart from the owner's public/private key, other access keys. Keys are kept in PKCS12 format. Keys have no limit of expiration.

The user keystore is protected by an access key. The access key is derived according to PKCS 5 specifications and is used to encrypt the private parts of the keystore. If the system has been EFS-enabled, keystores are automatically created for all already existing users when they log in for the first time.

If you migrate from a previous version to AIX 6, the keystores are automatically created when the users log in for the first time.

If a user does not need access to an encrypted file system, and you do not want to create a user keystore, you have to modify the corresponding user stanza in the /etc/security/user file.

2.5.3 Keystore content

Keystores contain both public and private information.

Public information kept in the keystore includes:

► User name.

► uid.

► Date of the last password modification.

► Keystore administrative mode: Can be either root guard or root admin. If root guard, it prevents root for changing root guard mode to root admin.

► Self cookie: Keystore access key encrypted by the user public key.

► The user public key is signed with a x509 self-signed certificate.

► If keystore admin mode is root admin, an admin cookie exists.

Private information kept in the keystore includes:

► A user private key. It also contains user deprecated keys.

► Group access keys for all groups the user belongs to. When the user keystore is opened, group access keys provide access to group keystores and group keys are loaded in the kernel.

► If the keystore belongs to root, the access key for efs_admin keystore exists.

The information in the keystore is organized in bags that can be either private or public. Public bags contain public information. Access to public bags has no restrictions.However, an access key is required to access private bags. The access is facilitated by using CliC objects.

The most important components of a keystore are the keys. Examples of private keys from user keystore are shown in Example 2-25 on page 101 and the meaning of their parameters is as follows:

Kind   This field displays if the key belongs to a user, a group or an administrator.

Id   This field displays the ID of the user or group that owns the key.

Type   This fields shows if the key is public or private.

Algorithm   This field shows that RSA algorithm is being used for the key and the key length in bits.

Validity   This filed shows if a key is still valid or if was deprecated.

Fingerprint   This field displays a number which, as the name implies, uniquely identifies any key.

Example 2-25   Private key examples from user keystore

```
Key #0:
    Kind .................... User key
    Id   (uid / gid) ........ 0
    Type .................... Private key
    Algorithm ............... RSA_1024
    Validity ................ Key is deprecated
    Fingerprint ..............
569ae24b:c2da3415:6e7e651e:b7d2f29f:1dda5ab9
 Key #1:
    Kind .................... User key
    Id   (uid / gid) ........ 0
    Type .................... Private key
    Algorithm ............... RSA_1024
    Validity ................ Key is valid
    Fingerprint ..............
f72bfdaa:5bbd8f93:3c021f29:177eae6a:e7e884b7
 Key #2:
    Kind .................... Group key
    Id   (uid / gid) ........ 7
    Type .................... Private key
    Algorithm ............... RSA_1024
    Validity ................ Key is valid
    Fingerprint ..............
648b67f3:f1f89672:14dcc664:231d0fe2:026a6314
 Key #3:
    Kind .................... Group key
    Id   (uid / gid) ........ 203
    Type .................... Private key
    Algorithm ............... RSA_1024
    Validity ................ Key is deprecated
    Fingerprint ..............
4a73c007:bb0e6b35:abe672c6:cdaf2964:5dd69586
Key #4:
    Kind .................... Admin key
    Id   (uid / gid) ........ 0
    Type .................... Private key
    Algorithm ............... RSA_1024
    Validity ................ Key is valid
    Fingerprint ..............
b8515c35:0829f5c9:f74fc031:3fac0195:8ab2ad99
```

Keystore content can be displayed using the `efskeymgr -v` command. When running in root guard mode, all operations pending on the keystore are displayed and you are prompted for action. The output will be similar to the one displayed in Example 2-26 on page 103, as described below:

► There are four pending operations on the user3 keystore. The keystore owner has been granted access to group1 keystore and user2 keystore. The keystore owner has also had access to user1 keystore revoked and is informed that the private key must be regenerated. user2 is prompted to accept the cookies.

► The keystore owner displays the uid of the owner. This keystore is owned by the user3 whose uid is 209.

► The keystore mode displays the keystore mode, either admin or guard. user3 keystore mode is root guard.

► The date the password was last changed.

► The algorithm, fingerprint, and validity of the private key of the keystore owner.

► Access keys that have been previously sent to the user3 keystore and already accepted. user3 has already accepted access cookies from user1 and group2.

► Access keys that have been previously sent to the user3 keystore in the form ok access cookies and not yet accepted. user3 has not accepted yet access cookies from user2 and group1. user3 has also received a regeneration cookie for private key regeneration and a remove cookie for revoking access to user1 keystore.

Example 2-26 Sample content of user keystore

```
$ efskeymgr -v
user3's EFS password:

The following operation(s) is(are) pending on your EFS keystore:
        You are granted access to group/group1 keystore.
Do you want to process this action now (y), later (n: default), or
never (d)? [ y / n / d ]n
        You are granted access to user/user2 keystore.
Do you want to process this action now (y), later (n: default), or
never (d)? [ y / n / d ]n
        You are removed access to user/user1 keystore.
Do you want to process this action now (y), later (n: default), or
never (d)? [ y / n / d ]n
        Your private key must be regenerated.
Do you want to process this action now (y), later (n: default), or
never (d)? [ y / n / d ]n
 Keystore content:
     Keystore owner ............ : uid 209
     Keystore mode ............. : guard: not managed by EFS
administrator
     Password changed last on .. : 05/15/07 at 18:09:58
    Private key:
          Algorithm : RSA_1024
        Fingerprint : 6c921f21:b10c9740:29acee26:4b5dd72b:23ea0849
           Validity : This key is valid.
    Access key to keystore user/user1
    Access key to keystore group/group2
Cookie:
  Type ...... : Regeneration (renew private key)
  Keystore .. : -
Cookie:
  Type ...... : Remove (removes access to)
  Keystore .. : user/user1
Cookie:
  Type ...... : Access (grants access to)
  Keystore .. : user/user2
Cookie:
  Type ...... : Access (grants access to)
  Keystore .. : group/group1
```

2.5.4 Keystore operations

Operations that modify a keystore include:

► Changing algorithm

► Changing key length

► Generating a new key

► Deleting a deprecated key

► Adding/removing a key to/from keystore

► Granting/revoking access to keystore

► Adding/removing the user to/from a group

At any time you can use the `efskeymgr -m` command to list the pending operations for the keystore. Example 2-27 shows four cookies' keys that will prompt for confirmation the first time the user open the keystore as follows:

► The first cookie has the type Regeneration and prompts the user for private key regeneration.

► The second cookie has the type Remove and prompts the user to remove the access key for the user1 keystore.

► The third cookie has the type Access the prompts the user to accept an access key for the user2 keystore.

► The fourth cookie has the type Access and prompts the user to accept an access key for group1 keystore.

Example 2-27   Pending operations on the keystore

```
$ efskeymgr -m
Cookie:
  Type ...... : Regeneration (renew private key)
  Keystore .. : -
Cookie:
  Type ...... : Remove (removes access to)
  Keystore .. : user/user1
Cookie:
  Type ...... : Access (grants access to)
  Keystore .. : user/user2
Cookie:
  Type ...... : Access (grants access to)
  Keystore .. : group/group1
```

2.5.5 Keystore operations

User keystores permit two modes of operation: root admin mode and root guard mode. Depending on the mode of operation, some operations are allowed or prohibited.

Root admin mode

Root admin mode is the default mode of operation. In this mode, there are some operations that are permitted, such as:

► Root can get access to the user keystore.

► Root can get access to the group keystore.

► Root can reset the user keystore password.

► Root can reset the group access key.

A consequence of root being able to get access to user keystore is that root can access user files.

Root guard mode

Root guard mode is the default mode of operation. In this mode, there are some operations that are not permitted, such as:

► Root cannot get access to the user keystore.

► Root cannot get access to the group keystore.

► Root cannot reset the user keystore password.

► Root cannot reset the group access key.

This mode of operation offers protection against a malicious root. It means that if the system is hacked and the hacker somehow manages to get root privilege, the hacker cannot have access to user or group keystores and therefore cannot have access to user files.

On the other hand, if the user loses his or her password root can cannot reset it. It means that no one can have access to user or group keystore and the information from user files can no longer be decrypted.

2.5.6 Changing the user keystore password

The keystore password is distinct from the login password.

You can have a separate password for the keystore. However, if the login password is identical to the keystore password, the user keystore is automatically opened and data from the keystore is automatically pushed into the kernel. If the passwords are not identical, the user must explicitly open the keystore and load the access keys in the kernel.

A user that does not have a keystore is still allowed to log in using the login password. The user will operate successfully if he or she does not access any data that is stored in an encrypted file system.

It is very important to understand that running the `password` command changes only login password and not the keystore password.

Any user can change his or her keystore password using the `efskeymgr -n` command. When running in root admin mode, both root and the user can change the user keystore password. Root changing a user keystore password is shown in Example 2-28.

Example 2-28   Changing a user password when running in root admin mode

```
# efskeymgr -k user/user3 -n
Enter new password for the keystore:
Enter the same password again:
```

2.5.7 Granting access to the user keystore

When running in root admin mode, there may be circumstances in which root decides to give a user access to the keystore of another user. A regular user cannot give access at his or her own keystore to another user.

In Example 2-29 on page 108, we show how the content of keystore changes when access to another user keystore is granted. We also discuss the effects of such operations in a more complex practical scenario. In Example 2-29 on page 108, the following occurs:

► Run the `efsmgr -v` command to display the content of the user3 keystore. He or she has access only at his or her own keystore.

► Root grants user3 access to the keystore of user1.

► Run the `efsmgr -v` command to display the content of the user3 keystore. The access key to the user1 keystore has been added.

- The `lsuser` command shows that user1 is also a member of group1 and group2.

- The content of the user1 keystore is loaded in the shell process using the `efskeymgr -o` command.

- The `efskeymgr -V` command shows the access key loaded in the current shell:

    - Access key for the user3 keystore

    - Access key for the user1 keystore

    - Access key for the keystore of the group with gid 202

    - Access key for the keystore of the group with gid 203

    user3 has access at his or her own key and all keys that are accessible from the user1 keystore.

- Root grants user5 access to the keystore of user3. If running in user mode, root would not be allowed to grant access to user keystore and would receive an error message similar to that displayed below:

    ```
    # efskeymgr -k user/user1 -s user/user3
    Encryption framework returned an error: (libefs bad parameter)
    Unable to get the key to be sent
    ```

- user5 logs in and run the `efsmgr -v` command to display the content of his or her keystore. The access key to user3 keystore has been added.

- user5 loads the content of his or her keystore in the current shell. The `efskeymgr -V` command shows the access key loaded in current shell:

    - Access key for the user3 keystore

    - Access key for the user1 keystore

    - Access key for the user5 keystore

    - Access key for the keystore of the group with gid 202

    - Access key for the keystore of the group with gid 203

    user5 has access at his or her own key and all keys that are accessible from user3 keystore, which in turn includes the all keys that are accessible from user1 keystore.

Example 2-29   Granting access to user keystore

```
$ id
uid=205(user3) gid=1(staff)
$ efskeymgr -v
 Keystore content:
      Keystore owner ............ : uid 205
      Keystore mode ............. : admin: managed by EFS administrator
      Password changed last on .. : 05/10/07 at 09:41:20
    Private key:
          Algorithm : RSA_1024
        Fingerprint : 30412121:e5a7b90d:dba7dd19:2c45b1e0:c331c09f
           Validity : This key is valid.
# efskeymgr -k user/user1 -s user/user3
root's EFS password:
$ id
uid=205(user3) gid=1(staff)
$ efskeymgr -v
 Keystore content:
      Keystore owner ............ : uid 205
      Keystore mode ............. : admin: managed by EFS administrator
      Password changed last on .. : 05/10/07 at 09:41:20
    Private key:
          Algorithm : RSA_1024
        Fingerprint : 30412121:e5a7b90d:dba7dd19:2c45b1e0:c331c09f
           Validity : This key is valid.
    Access key to keystore user/user1
$ lsuser user1
user1 id=203 pgrp=staff groups=staff,group1,group2 home=/home/user1
shell=/usr/bin/ksh
$ efskeymgr -o ksh
$ efskeymgr -V
List of keys loaded in the current process:
 Key #0:
    Kind .................... User key
    Id   (uid / gid) ........ 203
    Type .................... Private key
    Algorithm ............... RSA_1024
    Validity ................ Key is valid
    Fingerprint .............
eb1aab3e:39c3191c:15cb36d6:57bb2a7c:b3c6d356
 Key #1:
    Kind .................... User key
    Id   (uid / gid) ........ 205
    Type .................... Private key
```

```
      Algorithm ............... RSA_1024
      Validity ................ Key is valid
      Fingerprint .............
30412121:e5a7b90d:dba7dd19:2c45b1e0:c331c09f
 Key #2:
      Kind .................... Group key
      Id   (uid / gid) ........ 202
      Type .................... Private key
      Algorithm ............... RSA_1024
      Validity ................ Key is valid
      Fingerprint .............
19f16934:20a54e8e:d59aea33:111a37bf:06261785
 Key #3:
      Kind .................... Group key
      Id   (uid / gid) ........ 203
      Type .................... Private key
      Algorithm ............... RSA_1024
      Validity ................ Key is valid
      Fingerprint .............
bdf38da7:57cd4486:6794c1bf:5329e0c4:4d042fcc
# efskeymgr -k user/user3 -s user/user5
root's EFS password:
$ id
uid=207(user5) gid=1(staff)
$ efskeymgr -v
 Keystore content:
      Keystore owner ............ : uid 207
      Keystore mode ............. : admin: managed by EFS administrator
      Password changed last on .. : 05/10/07 at 11:41:51
    Private key:
           Algorithm : RSA_1024
         Fingerprint : 6a9423b3:f59f0497:2f0f8ba0:9805a358:e18b16cd
            Validity : This key is valid.
    Access key to keystore user/user3
$ efskeymgr -o ksh
$ efskeymgr -V
List of keys loaded in the current process:
 Key #0:
      Kind .................... User key
      Id   (uid / gid) ........ 203
      Type .................... Private key
      Algorithm ............... RSA_1024
      Validity ................ Key is valid
      Fingerprint .............
eb1aab3e:39c3191c:15cb36d6:57bb2a7c:b3c6d356
```

```
 Key #1:
   Kind ................... User key
   Id   (uid / gid) ........ 205
   Type ................... Private key
   Algorithm ............... RSA_1024
   Validity ................ Key is valid
   Fingerprint .............
30412121:e5a7b90d:dba7dd19:2c45b1e0:c331c09f
 Key #2:
   Kind ................... User key
   Id   (uid / gid) ........ 207
   Type ................... Private key
   Algorithm ............... RSA_1024
   Validity ................ Key is valid
   Fingerprint .............
6a9423b3:f59f0497:2f0f8ba0:9805a358:e18b16cd
 Key #3:
   Kind ................... Group key
   Id   (uid / gid) ........ 202
   Type ................... Private key
   Algorithm ............... RSA_1024
   Validity ................ Key is valid
   Fingerprint .............
19f16934:20a54e8e:d59aea33:111a37bf:06261785
 Key #4:
   Kind ................... Group key
   Id   (uid / gid) ........ 203
   Type ................... Private key
   Algorithm ............... RSA_1024
   Validity ................ Key is valid
   Fingerprint .............
bdf38da7:57cd4486:6794c1bf:5329e0c4:4d042fcc
```

2.5.8 Revoking access to user keystore

After root has granted access to a user keystore, he or she may decide to revoke this access. The scenario presented in this section should be read only after reading and understanding the scenario presented in 2.5.7, "Granting access to the user keystore" on page 106. We take the following steps shown in Example 2-30:

► Log in as user5 and run the **efskeymgr -V** command to show the access keys he or she has access to:

– Access key for the user3 keystore

– Access key for the user1 keystore

– Access key for the user5 keystore

– Access key for the keystore of the group with gid 202

– Access key for the keystore of the group with gid 203

► Log in as user3 and run the **efskeymgr -V** command to show the access keys he or she has access to:

– Access key for the user3 keystore

– Access key for the user1 keystore

– Access key for the keystore of the group with gid 202

– Access key for the keystore of the group with gid 203

► Root revokes user3's access to the keystore of user1 using the **efskeymgr** command.

► user3 runs the **efsmgr -v** command to display the content of the user3 keystore. He or she has now access only at his or her own keystore.

► user5 reloads in the current shell the content of his or her keystore. He or she has now access only to his or her own key and the access key of the user3 keystore. The access keys from the user1 keystore are no longer available.

Example 2-30   Revoking access to user keystore

```
$ id
uid=207(user5) gid=1(staff)
$ efskeymgr -V
List of keys loaded in the current process:
 Key #0:
   Kind ..................... User key
   Id   (uid / gid) ......... 203
   Type .................... Private key
   Algorithm ............... RSA_1024
```

```
      Validity ................ Key is valid
      Fingerprint ..............
eb1aab3e:39c3191c:15cb36d6:57bb2a7c:b3c6d356
 Key #1:
      Kind .................... User key
      Id   (uid / gid) ........ 205
      Type .................... Private key
      Algorithm ............... RSA_1024
      Validity ................ Key is valid
      Fingerprint ..............
30412121:e5a7b90d:dba7dd19:2c45b1e0:c331c09f
 Key #2:
      Kind .................... User key
      Id   (uid / gid) ........ 207
      Type .................... Private key
      Algorithm ............... RSA_1024
      Validity ................ Key is valid
      Fingerprint ..............
6a9423b3:f59f0497:2f0f8ba0:9805a358:e18b16cd
 Key #3:
      Kind .................... Group key
      Id   (uid / gid) ........ 202
      Type .................... Private key
      Algorithm ............... RSA_1024
      Validity ................ Key is valid
      Fingerprint ..............
19f16934:20a54e8e:d59aea33:111a37bf:06261785
 Key #4:
      Kind .................... Group key
      Id   (uid / gid) ........ 203
      Type .................... Private key
      Algorithm ............... RSA_1024
      Validity ................ Key is valid
      Fingerprint ..............
bdf38da7:57cd4486:6794c1bf:5329e0c4:4d042fcc
# efskeymgr -k user/user3 -S user/user1
root's EFS password:
$ id
uid=205(user3) gid=1(staff)
$ efskeymgr -v
 Keystore content:
      Keystore owner ............ : uid 205
      Keystore mode ............. : admin: managed by EFS administrator
      Password changed last on .. : 05/10/07 at 09:41:20
       Private key:
```

```
            Algorithm : RSA_1024
          Fingerprint : 30412121:e5a7b90d:dba7dd19:2c45b1e0:c331c09f
             Validity : This key is valid.
$ id
uid=207(user5) gid=1(staff)
$ efskeymgr -o ksh
$ efskeymgr -V
List of keys loaded in the current process:
 Key #0:
    Kind .................... User key
    Id   (uid / gid) ........ 205
    Type ................... Private key
    Algorithm ............... RSA_1024
    Validity ................ Key is valid
    Fingerprint .............
30412121:e5a7b90d:dba7dd19:2c45b1e0:c331c09f
 Key #1:
    Kind .................... User key
    Id   (uid / gid) ........ 207
    Type ................... Private key
    Algorithm ............... RSA_1024
    Validity ................ Key is valid
    Fingerprint .............
6a9423b3:f59f0497:2f0f8ba0:9805a358:e18b16cd
```

2.5.9 Accepting access keys

When running in root guard mode, you may be prompted to accept access keys
that might have been granted to you. The access keys are sent to the user
keystore in the form of cookies. You can chose to accept the access key, to
decline the accept the access key, or to make a decision later. If you choose to
make a decision later, you are prompted for decision every time you open your
keystore. The content of the keystore is updated according to your actions. A
typical scenario in which an access cookie ia being sent is shown in
Example 2-31 on page 114:

► user1 is prompted to accept access key when trying to open his or her own
  keystore.

► user1 accepts the access key for group1, refuses the access key for group2,
  and postpones the decision regarding the access key for the user3 keystore.

► The new content of the keystore is displayed, including the access cookies.

Example 2-31   User is prompted to accept access keys

```
$ id
uid=214(user1) gid=1(staff) groups=206(group1),207(group2)
$ efskeymgr -v
user1's EFS password:

The following operation(s) is(are) pending on your EFS keystore:
        You are granted access to group/group1 keystore.
Do you want to process this action now (y), later (n: default), or
never (d)? [ y / n / d ]y
        You are granted access to group/group2 keystore.
Do you want to process this action now (y), later (n: default), or
never (d)? [ y / n / d ]n
        You are granted access to user/user3 keystore.
Do you want to process this action now (y), later (n: default), or
never (d)? [ y / n / d ]n
 Keystore content:
     Keystore owner ............ : uid 214
     Keystore mode ............. : guard: not managed by EFS
administrator
     Password changed last on .. : 05/13/07 at 15:28:06
    Private key:
         Algorithm : RSA_1024
        Fingerprint : 55a4e75e:fe056880:8d305005:b2f817fa:3b9f2264
          Validity : This key is valid.
    Access key to keystore group/group1
Cookie:
  Type ...... : Access (grants access to)
  Keystore .. : user/user3
Cookie:
  Type ...... : Access (grants access to)
  Keystore .. : group/group2
```

2.5.10 Granting security credentials to a process

In Example 2-32, we offer an example of granting security credentials to a specific process, described as follows:

► Log in to the system as root. There are no keys loaded in the current process.

► Run the **efskeymgr -o ksh** command to create a new shell and load the keys into this process.

► Verify that the keys are indeed loaded in the new shell that has just ben created.

Example 2-32   Loading keys for a new shell

```
# efskeymgr -V
There is no key loaded in the current process.
# man efskeymgr
# efskeymgr -o ksh
root's EFS password:
# efskeymgr -V
List of keys loaded in the current process:
 Key #0:
   Kind .................... User key
   Id   (uid / gid) ........ 0
   Type ................... Private key
   Algorithm ............... RSA_1024
   Validity ................ Key is valid
   Fingerprint .............
59da6d2f:86c1295d:e6752cf5:73bde901:679ef770
 Key #1:
   Kind .................... Group key
   Id   (uid / gid) ........ 7
   Type ................... Private key
   Algorithm ............... RSA_1024
   Validity ................ Key is valid
   Fingerprint .............
973bc437:3e06bf57:a17b1fec:eb815727:950894e3
 Key #2:
   Kind .................... Admin key
   Id   (uid / gid) ........ 0
   Type ................... Private key
   Algorithm ............... RSA_1024
   Validity ................ Key is valid
   Fingerprint .............
32c9aff2:552a5a87:7e5bd819:e05d938e:21a077bf
```

2.5.11 Exporting the content of keystore

In Example 2-33, we offer an example of exporting the keystore in PKCS #12 format and importing it in OpenSSL as follows:

► Export the keystore into a file using the **efskeymgr** command. You will be prompted twice for a password that will protect this file.

► Run OpenSSL if it is installed on your system.

► Use the **pkcs12 -in /tmp/keyfile -info -nodes** subcommand to display the information contained in the keystore and avoid encryption of the private key.

Example 2-33   Exporting keystore in PKCS #12 format and importing it using OpenSSL

```
# efskeymgr -e /tmp/keyfile
Enter password for the new PKCS#12-protected file:
Enter the same password again:
# /usr/linux/bin/openssl
OpenSSL> pkcs12 -in /tmp/keyfile -info -nodes
Enter Import Password:
MAC Iteration 2000
MAC verified OK
PKCS7 Data
Certificate bag
Bag Attributes: <No Attributes>
subject=/CN=CLiC v4.0 0128D26E
issuer=/CN=CLiC v4.0 0128D26E
-----BEGIN CERTIFICATE-----
MIIBqjCCARMCBAEoOm4wCwYJKoZIhvcNAQEBMBOxGzAZBgNVBAMTEkNMaUMgdjQu
MCAwMTI4RDI2RTAeFwOwNjA1MDgwMDAwMDBaFwOxNjA1MDgwMDAwMDBaMBOxGzAZ
BgNVBAMTEkNMaUMgdjQuMCAwMTI4RDI2RTCBnzANBgkqhkiG9wOBAQEFAAOBjQAw
gYkCgYEAqx36ny+2jrpjE5SMlz+vOtldbub6928nEOvf5IdLz9qn+V8/OdwTSokr
zN9EnRHeflzuFMGy2Zrztd3yOlP2QJsfL3sQUT3l9Flf4NmTTTjg/ToWroduYm6O
cOoDPvxffNh4/YSN/J8JFXKGFi8U5zttObpMAQsNjiLXJbP3IfkCAwEAATANBgkq
hkiG9wOBAQUFAAOBgQB5soWOIQbccHj5Z8im5OZmqV6GGOhrtLAD9Qo/Wdu9hzVI
uzQ7eJ3a7yVkKWNaENT8nb522fNzvEefOdm+pkzOJrOEHCQQcksvEqjv8VHFt8oo
mTQXX5ZjyWcVG3p67LbyKHsiRWhVFp/siFAnRD9dXb6uZFNKzT3sD211pODh1Q==
-----END CERTIFICATE-----
Shrouded Keybag: pbeWithSHA1And3-KeyTripleDES-CBC, Iteration 2000
Bag Attributes: <No Attributes>
Key Attributes: <No Attributes>
-----BEGIN RSA PRIVATE KEY-----
MIICXAIBAAKBgQCrHfqfL7aOumMTlIyXP6862V1u5vr3bycQ69/khOvP2qf5Xz85
3BNKiSvM3OSdEd5+XO4UwbLZmvO13fI6U/ZAmx8vexBRPeXOWV/g2ZNNOOD9Ohau
```

```
h25ibo5w6gM+/F982Hj9hI38nwkVcoYWLxTnO2O5ukwBCw2OItcls/ch+QIDAQAB
AoGABxuKLEWHlYXGwBVKAlhKAi2YKhdHB7MNZ4Wx+Y89xhGaK4taGCBSNCTeywuK
Zo8g85UO6zwhHn/HWApdhDBg8zxPLlTNx4HcbAVnE9pV+o2nuLo8nRtWKHp16Rmc
dKw8JQZSE8BdLPRv24DXxxoJ+pj1y6pXzlwWFYoeSURbqNkCQQDWVR9dAoLrmgNB
ClLz/sSi7dTrwArgcdXkobl5UGmGh86frvpONGXjSEJhPtihZtvmVOLb4wC3nOXf
oat4xOY9AkEAzGId6Wkzr7QS53xCm6msFwdrRLLEOpzOzinCoRgPT11mjIJIUtYK
QRuY1hFP6LSlUUPaiADUcw8zlV8iLZEibQJACfcFMUbXnWqYsiJvvuZecBDMsHVK
TCsevbWDMQ+7CEUyJxxOiMRI5GtMosbQPZjRqoDR11VTiDXkPiR/G2tUJQJBAMhY
PZ35w/KuAAHuKpTQQ6LnYNOcqBcUSFx3TxP+s7NGiSme5CpfKfochTsCCWTA/3Sj
oOnEdny7IAOCF8y3AhkCQES0l8uLBDtGAng4SIqU/UOuogJtwaRdp+CnOG8So2WP
y9o6Hn7Q+mGOynsOiBV1mlTwVO6oAFZcU6BSQ4Ujx6c=
-----END RSA PRIVATE KEY-----
OpenSSL> quit
```

2.5.12 User private keys

User keys may be deprecated or deleted when they are no longer considered secure. We show how user private keys can be exported from the user keystore.We offer an in-depth view of keystore content.

In Example 2-34 on page 118, we offer an example of how private keys are created, deprecated, and deleted as follows:

► Run the **efskeymgr -v** command to display the current keystore content. It contains one 1024-bit RSA key.

► Export the current content of the keystore into keyfile1 using the **efskeymgr -e keyfile1** command.

► Verify the content of the keyfile1 using the OpenSSL subcommand **pkcs12 -in keyfile1 -info -nodes**. The 1024-bit RSA private key is displayed.

► Change the keystore private key to a 2048-bit RSA key using the **efskeymgr -R RSA_2048** command.

► Run the **efskeymgr -v** command to display the current keystore content. The new 2048-bit key has been activated and the 1024-bit key was deprecated.

► Export the current content of the keystore into keyfile2 using the **efskeymgr -e keyfile2** command.

► Verify the content of the keyfile2 using the OpenSSL subcommand **pkcs12 -in keyfile2 -info -nodes**. Both 1024-bit and 2048-bit RSA private keys are displayed.

► Change the keystore private key to a 4096-bit RSA key using the **efskeymgr -R RSA_4096** command.

► Run the **efskeymgr -v** command to display the current keystore content. The new 4096-bit key has been activated and the 2048-bit key was deprecated.

► Export the current content of the keystore into keyfile3 using the **efskeymgr -e keyfile3** command.

► Verify the content of the keyfile3 using the OpenSSL command **pkcs12 -in keyfile3 -info -nodes**. All three keys are displayed. Both deprecated keys are displayed.

► Delete the 2048-bit key using the **efskeymgr -D 3661bf34:530116eb:1861a3eb:0cf71b91:91ca25e9** command.

► Run the **efskeymgr -v** command to display the current keystore content. The new 2048-bit key has been deleted.

► Export the current content of the keystore into keyfile4 using the **efskeymgr -e keyfile4** command.

► Verify the content of the keyfile4 using the OpenSSL subcommand **pkcs12 -in keyfile4 -info -nodes**. The new 2048-bit key has been deleted.

Example 2-34   Creating, deprecating, and deleting private RSA keys

```
# efskeymgr -v
 Keystore content:
     Keystore owner ............ : uid 0
     Keystore mode ............. : admin: managed by EFS administrator
     Password changed last on .. : 04/18/07 at 04:08:43
   Private key:
         Algorithm : RSA_1024
       Fingerprint : 47dab395:99d8aa70:82659beb:700a4a6e:e232c8e4
          Validity : This key is valid.
   Access key to keystore group/security
   Access key to keystore admin/

# efskeymgr -e keyfile1
Enter password for the new PKCS#12-protected file:
Enter the same password again:

OpenSSL> pkcs12 -in keyfile1  -info -nodes
Enter Import Password:
MAC Iteration 2000
MAC verified OK
PKCS7 Data
Certificate bag
Bag Attributes: <No Attributes>
subject=/CN=CLiC v4.0 4613EBC8
issuer=/CN=CLiC v4.0 4613EBC8
```

```
-----BEGIN CERTIFICATE-----
MIIBqjCCARMCBEYT68gwCwYJKoZIhvcNAQEBMBOxGzAZBgNVBAMTEkNMaUMgdjQu
MCAONjEzRUJDODAeFwOwNjA1MDgwMDAwMDBaFwOxNjA1MDgwMDAwMDBaMBOxGzAZ
BgNVBAMTEkNMaUMgdjQuMCAONjEzRUJDODCBnzANBgkqhkiG9wOBAQEFAAOBjQAw
gYkCgYEAvMeyOsd1hh8YdoExhATlSR9KQP61zOQGQgp2c5aU97N3OrxCqpwAcm6L
dE4yzOl9WsQWRg8yJqkzE3J9G2547aq2vNxRuo/zgM5YZnK1JAnRZyZOONGDooVf
sb782Q9jL9n8sudFzEpIQ9jb68+kn7maXCGMHFImEnZjv3YGtOsCAwEAATANBgkq
hkiG9wOBAQUFAAOBgQAhFp2ujXe9la4/9c25loqONgBot8kiJoH7OBuXRC8UexoP
wOIp3Q17m8nyW3ymGcyfXtOUu2eC5NLWpr3W6orhr/ffwEVYJ6f7r5b4ADwBeHUL
3xYcQ4wnn/cO8E9NugmBM9ix5IyxPgZsy4VeNOsKY5YvtkNsWFX3vCqOUMA+Ig==
-----END CERTIFICATE-----
Shrouded Keybag: pbeWithSHA1And3-KeyTripleDES-CBC, Iteration 2000
Bag Attributes: <No Attributes>
Key Attributes: <No Attributes>
-----BEGIN RSA PRIVATE KEY-----
MIICXQIBAAKBgQC8x7LSx3WGHxh2gTGEBOVJHOpA/rXM5AZCCnZzlpT3s3fSvEKq
nABybotOTjLM6X1axBZGDzImqTMTcnObbnjtqra83FG6j/OAzlhmcrUkCdFnJk7Q
OYOihV+xvvzZD2Mv2fyy5OXMSkhD2Nvrz6SfuZpcIYwcUiYSdmO/dga3SwIDAQAB
AoGAHJ7KtyxYiSQiEXv7wE/9B616lJ5TOgOItAQhujDMVpNlvqlzvVRCYxbsWzks
eCbhWNydTd4plHVBfO4gwS2/SnHGwnQwO6LcVgkqONSDqbf2yTJGBUDUwmZOqOxp
HPrFsl99VzrJM+/pgEQcqEbdwc7l82NgCBcuPYtNpNSKL9ECQQD6X5os5wTbWdzo
ukQlWTmKWpdn3K+muDwIwy2oGesSY2G24i5IZcrrsX5gzUlSI4Z22zG9m1JnaItC
/b44Tl9bAkEAwQW/8OG4Vd216ULbLNoasoA1GVGfIiVJCzqzEO2nt19bQwwNwwrt
zmzJVczCi8cEWSFQxbe/3geOAYiNfoT6OQJBAO+1/IylGndQODpZq8j+ZiLanT8p
gVrj5UaWZ+4b8n6GfBV2880I+IEOTMUthteHf8PoFPVt8jVjWIHpIugROSUCQBUp
XTa6eGwph3UQWFkdfEnPloK1GR4OOkLZ56HWfEm1UZsTKjsU6qdz88rNTRLn+ckP
xvw2PfnImEAAyYpyZ9ECQQDSx4/ZF2/Uk/lOBe5SOBkYKKDnVrioXBjLqn161ERa
8ABuiz3EeI5knBx/sd8FVhNF+Izka5qcA4rd7XYvar1s
-----END RSA PRIVATE KEY-----

# efskeymgr -R RSA_2048

# efskeymgr -v
 Keystore content:
     Keystore owner ............ : uid 0
     Keystore mode ............. : admin: managed by EFS administrator
     Password changed last on .. : 04/18/07 at 04:08:43
    Private key:
         Algorithm : RSA_2048
        Fingerprint : 3661bf34:530116eb:1861a3eb:0cf71b91:91ca25e9
          Validity : This key is valid.
    Private key:
         Algorithm : RSA_1024
        Fingerprint : 47dab395:99d8aa70:82659beb:700a4a6e:e232c8e4
           Validity : This key was deprecated on 04/20/07 at 07:35:45.
```

```
        Access key to keystore group/security
        Access key to keystore admin/

# efskeymgr -e keyfile2
Enter password for the new PKCS#12-protected file:
Enter the same password again:


OpenSSL> pkcs12 -in keyfile2 -info -nodes
Enter Import Password:
MAC Iteration 2000
MAC verified OK
PKCS7 Data
Certificate bag
Bag Attributes: <No Attributes>
subject=/CN=CLiC v4.0 7F14828C
issuer=/CN=CLiC v4.0 7F14828C
-----BEGIN CERTIFICATE-----
MIICrzCCAZcCBH8UgowwCwYJKoZIhvcNAQEBMBOxGzAZBgNVBAMTEkNMaUMgdjQu
MCA3RjE0ODI4QzAeFw0wNjA1MDgwMDAwMDBaFw0xNjA1MDgwMDAwMDBaMBOxGzAZ
BgNVBAMTEkNMaUMgdjQuMCA3RjE0ODI4QzCCASIwDQYJKoZIhvcNAQEBBQADggEP
ADCCAQoCggEBAKTOYsK3m2dGTShLsYfKGwbJiGDvySziwkTQmiTd158Pkg4bNSKS
Jyrdgp77zZoO7Es19377WgjJGOLW6GGpJmTOzp8yZBOJxdwLJ6nxdtOo2S6uhtb3
zbXpEuMdf1OisPaztjORl1GlepT5Qz44el4LCqgJXcHqRR7ql9oRLhGOD1V6vzYR
hE1LuVdxuRFJoetDiTlpJzwu3OkXAVAsrRIIhJzt5apewv3Li6JVynGUX4EYSbRz
Xl+GnoQdChsFEG/ZZyA1Nlply3VJR8gb1z5SuguCPE4aG/vHpIfq/bl9B4L8OFvh
3VA4usZ45oz2kLwGTkUnpBxAk0Iv3GJ1Tf8CAwEAATANBgkqhkiG9w0BAQUFAAOC
AQEAdkwLxKnHtsJYwNqu62G5/d8FzAvEC2ky3RkPVjdGfySvS5LpKcEwc+DoZMWh
Ripey9OuX/D4BlY/Qin+5vqiFD+MLA3w9yaHHsPaTZMczgaqGSyCTx+Z8GBdEdpQ
LybZCyI3NYNlibBAcYZ/g/jAn7SuOrxJwJgopJseGfqoqvfht/wA1ccBtmfBZWZO
CklxPErGVrkMFAuwl4PUPvZU/vpvV9yVUuDa/VL8ahxmCMby/WeQZJWNHZ8RJqi7
IO+JeQWtZ6uGoNTZUJdxvuwefYJts7vw6lx92j4JYDnWzVB8/kZHaa8QumZAU/pR
buDin1FTUmyt6cQOeAnfaO8FMQ==
-----END CERTIFICATE-----
Shrouded Keybag: pbeWithSHA1And3-KeyTripleDES-CBC, Iteration 2000
Bag Attributes: <No Attributes>
Key Attributes: <No Attributes>
-----BEGIN RSA PRIVATE KEY-----
MIICXQIBAAKBgQC8x7LSx3WGHxh2gTGEBOVJHOpA/rXM5AZCCnZzlpT3s3fSvEKq
nABybotOTjLM6X1axBZGDzImqTMTcnObbnjtqra83FG6j/OAzlhmcrUkCdFnJk7Q
0YOihV+xvvzZD2Mv2fyy5OXMSkhD2Nvrz6SfuZpcIYwcUiYSdmO/dga3SwIDAQAB
AoGAHJ7KtyxYiSQiEXv7wE/9B616lJ5TOgOItAQhujDMVpNlvqlzvVRCYxbsWzks
eCbhWNydTd4plHVBfO4gwS2/SnHGwnQwO6LcVgkqONSDqbf2yTJGBUDUwmZOqOxp
HPrFsl99VzrJM+/pgEQcqEbdwc7l82NgCBcuPYtNpNSKL9ECQQD6X5os5wTbWdzo
ukQlWTmKWpdn3K+muDwIwy2oGesSY2G24i5IZcrrsX5gzUlSI4Z22zG9m1JnaItC
/b44Tl9bAkEAwQW/8OG4Vd216ULbLNoasoA1GVGfIiVJCzqzEO2nt19bQwwNwwrt
```

zmzJVczCi8cEWSFQxbe/3geOAYiNfoT6OQJBAO+1/IylGndQODpZq8j+ZiLanT8p
gVrj5UaWZ+4b8n6GfBV2880I+IEOTMUthteHf8PoFPVt8jVjWIHpIugROSUCQBUp
XTa6eGwph3UQWFkdfEnPloK1GR4OOkLZ56HWfEm1UZsTKjsU6qdz88rNTRLn+ckP
xvw2PfnImEAAyYpyZ9ECQQDSx4/ZF2/Uk/lOBe5SOBkYKKDnVrioXBjLqn161ERa
8ABuiz3EeI5knBx/sd8FVhNF+Izka5qcA4rd7XYvar1s
-----END RSA PRIVATE KEY-----
Shrouded Keybag: pbeWithSHA1And3-KeyTripleDES-CBC, Iteration 2000
Bag Attributes: <No Attributes>
Key Attributes: <No Attributes>
-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEApPRiwrebZOZNKEuxh8obBsmIYO/JLOLCRNCaJN3Xnw+SDhs1
IpInKt2CnvvNmjTsSzX3fvtaCMkY4tboYakmZPTOnzJkE4nF3AsnqfF2O6jZLq6G
1vfNtekS4x1/XSKw9rO2PRGXUaV6lPlDPjh6XgsKqAldwepFHuqX2hEuEbQPVXq/
NhGETUu5V3G5EUmh6OOJOWknPC7c6RcBUCytEgiEnO3lql7C/cuLolXKcZRfgRhJ
tHNeX4aehBOKGwUQb9lnIDU2WmXLdUlHyBvXPlK6C4I8Thob+8ekh+r9uXOHgvw4
W+HdUDi6xnjmjPaQvAZORSekHECTQi/cYnVN/wIDAQABAoIBAAE2WEUQ1NJCalGB
ig+gTizxZORkVMqQl1qCvRHCsTgBSVWMrCDaSr7GvzkBq6zX63AOYE68PldqdFr9
LmkBJzz+b8UP7LjlA97A7M+zDqQx+tucIDcqL5eNOGxXw/+58IU58RDK/G/MPc38
sz+LuenJmXmxseIf+bPvzBld9I5GmOobm2X5JEZHXqHHhwAdwpFiD5hPq/CmUOjO
xVO4JKfsfjL5eLM9QPYcgZ439pz5kfsbYxuuaZfUY2iYScnamar+XrwbPh79KmsE
+9Blc8PCFhdPndspr1eBahykIaY2EBoUvJLqgEQA1H48+Fce5+aI1ATV7iBzB+lK
gwtjoKECgYEA2iA35ZbC2Zqz8NWqJ3OuOhg8a7KP2zLS/OXw1LLIwU8dIfY4Bled
IhxSizEtJu29XeTvLEZIDBIhDEnxXOaVXhU8NAjfKJcG7ZInMRc6BnKE2Jz4ClyD
UOOvtwTyX27ZSTdCRCneQNxYt8WnZOGN/dlrIPP2UPfC12qZv85FtDkCgYEAwZiv
clSVu/FOB8WuZcu8xaZIYosvAjXAbRFsvRjMNf7swtGworslcjV5hgV+/JgIyooq
TMT6VefyY8475AFGfiyrOx5RHSL5G6IcTo7CW+e2B+clONdmKPafXxuKGS8hXkN8
NdCpBw3VNk+gNJZiIBGhNrt5mFInIUXGN95Vw/cCgYB4NFEZgN+YM5z+F4FRhHrO
5PRHFXwZfASzEMQLMwtXHFYdpSZyuE2rwpfqhQsw7Mryt3rmlOSVAlicARO18oxV
8LXgpxLzt2bl/SkD8JzVeL6k8LvwWdM1S5PF9D/tP5UWuBEYpOoHzGZHv/Djszkg
r3ROEKOpMyB4vhJElVPsYQKBgCXydcFlTwxN4b58Qxj0/IWJeHrIaWh3fMZkOfo+
2BhlZacyTvs2z7o5PR7GCQQeEOsSdgQiQCeClYJ2tEqW7Whh/TQepyuc5VNDTWUg
mnxFSOr3pfPvpLi2zrpnvYP+Nv4xIDGOOs2FKpvAs9ha+dTrX7xItybiQ4VEFfxx
ekI1AoGBAKaMJPWH52UpgAzgW3CotoFv25kAcTnL4vJ6QK5SPKxWPRyduGaMnPgv
GawZhiCGM921ui/Zq521Keohik5Ak2flLtgMnnuzV6W2Nc7nfVqUqD7ekKhCPFog
C6t2lfrwq1aq2iZ6pkO4jpk7dsbeuUDHLoQIzZB3HVRrVhXfu/Ag
-----END RSA PRIVATE KEY-----

```
# efskeymgr -v
 Keystore content:
     Keystore owner ............ : uid 0
     Keystore mode ............. : admin: managed by EFS administrator
     Password changed last on .. : 04/18/07 at 04:08:43
    Private key:
         Algorithm : RSA_4096
        Fingerprint : f961c000:8305d1ba:65a72c48:eeafde52:b954bf78
```

```
                    Validity : This key is valid.
        Private key:
              Algorithm : RSA_2048
            Fingerprint : 3661bf34:530116eb:1861a3eb:0cf71b91:91ca25e9
               Validity : This key was deprecated on 04/20/07 at 08:08:44.
        Private key:
              Algorithm : RSA_1024
            Fingerprint : 47dab395:99d8aa70:82659beb:700a4a6e:e232c8e4
               Validity : This key was deprecated on 04/20/07 at 07:35:45.
        Access key to keystore group/security
        Access key to keystore admin/
# efskeymgr -e keyfile3
Enter password for the new PKCS#12-protected file:
Enter the same password again:
#
OpenSSL> pkcs12 -in keyfile3 -info -nodes
Enter Import Password:
MAC Iteration 2000
MAC verified OK
PKCS7 Data
Certificate bag
Bag Attributes: <No Attributes>
subject=/CN=CLiC v4.0 17162B54
issuer=/CN=CLiC v4.0 17162B54
-----BEGIN CERTIFICATE-----
MIIErzCCApcCBBcWK1QwCwYJKoZIhvcNAQEBMBOxGzAZBgNVBAMTEkNMaUMgdjQu
MCAxNzE2MkI1NDAeFwOwNjA1MDgwMDAwMDBaFwOxNjA1MDgwMDAwMDBaMBOxGzAZ
BgNVBAMTEkNMaUMgdjQuMCAxNzE2MkI1NDCCAiIwDQYJKoZIhvcNAQEBBQADggIP
ADCCAgoCggIBAOqqO3DViNm5C326mJdEmM1w/17RXUBxo7pPhAZxMnpOo+3Ku+gO
W8YQC22o8dxSGaEFUfm6jjPVyJyEb5fWNpDqk1huLdfM/uTvO4O1TZNzPclHZXEQ
NpIxtgo1i3t1OGm+8DahvdfnILhWlrarvAYqOSbxelHcbnU6Ppr5kIfZez1xwGCv
3JWfQuvhf2y4JSwb7V8O36yuEwnS7OmdGQl6itMtqZOw9hTyoRDaXNZM5G3MVCum
xQqxGeTjRyaLnbB2vs9z1aJ3brMM2jnJnph/Re/3K2xfn4idv/hKW3MhP2AakLIi
cOo1yhgHM7oy6vTGd4x/zkncbPD2FuxqQCgtso9OVd69A6x66uDPs+VbN/EXeW2J
yipClUr9Bclu1ZRxLaf9huWMQ/uQLXMChN9GEIhq+/nBfEkfLjdSSFdVONeWAzoR
+Og9plOR/VV2ZhO/Yf2cZfKrNZ/pa5BvHeMzGBZG+mj4lpOnneaV8q2ZFX2XzaZ1
J1D1YsN3DRihz/uUu2vHfzDdSiCheALgn6MPk58mt5unRN7STWZJGeOQZbWG3Ijo
O9HDXlDSnt3vzPoBOyFxhOYBPO+QxejHExhF9F2/HHJnzu+zk67dAxzVV/xfnoJl
VZ42hUbb3zRDWApxx9II9DWCHtmqpjrklSltJ/XAIENdz4nOuoAIK1G3AgMBAAEw
DQYJKoZIhvcNAQEFBQADggIBALr+B3OWQPl8cGvP/ZPYV7KLyKH+TCQwRPp/WOye
wbDDrhwcVsEKeYhtFDcifVsXdM4FGbC14Fbv7UEQpWOhQO1IHHUOutXEyFe1O7wh
SItLBJJCKjvHPpwyDmey3pe/JsdKWt3hWWD8e7TwjcBY9lp7Z3fWH11CTnI8VA+3
12mOY6DsBTwa1UfFoYR4Fl2GBzYvN8j3wXpdPGznvd4s/VHcopcu/ya6aEU+ZMOq
ehAnlfEDw9ASVHYzt77+XxskGicRV+OTe5X/pt4ja89MjRY/nXFO46cnxTivH2eI
xI5Z47yFdyGVcsfWkhdIyV69XZ+WN29G7z143QyUWA1FFFq9vjF4KPERou5nO3HU
```

floFhRamhJDKyuZBHCcC4fw1CcXUK19vRUps1i1wM/oC9Iwd4h8oj+PFN/+tR53u
rbdYsMAP71duHon2AWTeaZlgRDx9Ts9vqywOoO5C/OXXEnfKuF+AfDuv3jxHh84M
jRkEBOLasqZWrEDIMUoXyi7ypICQ5NOhroOPhw7QQjaJb7CeDtCzCtOwosqQODdL
87OmEWkNLY2pvg8T6VgqepNpqNIrKw4y6ILViyqPLT6/lnri7c7p5qm8XKgkyMdl
rPpS/9ZtxfujJSjjIAQrNKjoHhewrBBOgq2JZ37PU+BB/TOmjWOdODX4q5QJOcXG
TUk7
-----END CERTIFICATE-----
Shrouded Keybag: pbeWithSHA1And3-KeyTripleDES-CBC, Iteration 2000
Bag Attributes: <No Attributes>
Key Attributes: <No Attributes>
-----BEGIN RSA PRIVATE KEY-----
MIICXQIBAAKBgQC8x7LSx3WGHxh2gTGEBOVJHOpA/rXM5AZCCnZzlpT3s3fSvEKq
nABybotOTjLM6X1axBZGDzImqTMTcnObbnjtqra83FG6j/OAzlhmcrUkCdFnJk7Q
OYOihV+xvvzZD2Mv2fyy50XMSkhD2Nvrz6SfuZpcIYwcUiYSdmO/dga3SwIDAQAB
AoGAHJ7KtyxYiSQiEXv7wE/9B616lJ5TOgOItAQhujDMVpNlvqlzvVRCYxbsWzks
eCbhWNydTd4plHVBfO4gwS2/SnHGwnQwO6LcVgkqONSDqbf2yTJGBUDUwmZOqOxp
HPrFsl99VzrJM+/pgEQcqEbdwc7l82NgCBcuPYtNpNSKL9ECQQD6X5os5wTbWdzo
ukQlWTmKWpdn3K+muDwIwy2oGesSY2G24i5IZcrrsX5gzUlSI4Z22zG9m1JnaItC
/b44Tl9bAkEAwQW/8OG4Vd216ULbLNoasoA1GVGfIiVJCzqzEO2nt19bQwwNwwrt
zmzJVczCi8cEWSFQxbe/3geOAYiNfoT6OQJBAO+1/IylGndQODpZq8j+ZiLanT8p
gVrj5UaWZ+4b8n6GfBV2880I+IEOTMUthteHf8PoFPVt8jVjWIHpIugROSUCQBUp
XTa6eGwph3UQWFkdfEnPloK1GR4OOkLZ56HWfEm1UZsTKjsU6qdz88rNTRLn+ckP
xvw2PfnImEAAyYpyZ9ECQQDSx4/ZF2/Uk/lOBe5SOBkYKKDnVrioXBjLqn161ERa
8ABuiz3EeI5knBx/sd8FVhNF+Izka5qcA4rd7XYvar1s
-----END RSA PRIVATE KEY-----
Shrouded Keybag: pbeWithSHA1And3-KeyTripleDES-CBC, Iteration 2000
Bag Attributes: <No Attributes>
Key Attributes: <No Attributes>
-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEApPRiwrebZOZNKEuxh8obBsmIYO/JLOLCRNCaJN3Xnw+SDhs1
IpInKt2CnvvNmjTsSzX3fvtaCMkY4tboYakmZPTOnzJkE4nF3AsnqfF2O6jZLq6G
1vfNtekS4x1/XSKw9rO2PRGXUaV6lPlDPjh6XgsKqAldwepFHuqX2hEuEbQPVXq/
NhGETUu5V3G5EUmh6OOJOWknPC7c6RcBUCytEgiEnO3lql7C/cuLolXKcZRfgRhJ
tHNeX4aehBOKGwUQb9lnIDU2WmXLdUlHyBvXPlK6C4I8Thob+8ekh+r9uXOHgvw4
W+HdUDi6xnjmjPaQvAZORSekHECTQi/cYnVN/wIDAQABAoIBAAE2WEUQ1NJCalGB
ig+gTizxZORkVMqQl1qCvRHCsTgBSVWMrCDaSr7GvzkBq6zX63AOYE68PldqdFr9
LmkBJzz+b8UP7LjlA97A7M+zDqQx+tucIDcqL5eNOGxXw/+58IU58RDK/G/MPc38
sz+LuenJmXmxseIf+bPvzBld9I5GmOobm2X5JEZHXqHHhwAdwpFiD5hPq/CmUOjO
xVO4JKfsfjL5eLM9QPYcgZ439pz5kfsbYxuuaZfUY2iYScnamar+XrwbPh79KmsE
+9Blc8PCFhdPndspr1eBahykIaY2EBoUvJLqgEQA1H48+Fce5+aI1ATV7iBzB+lK
gwtjoKECgYEA2iA35ZbC2Zqz8NWqJ3OuOhg8a7KP2zLS/OXw1LLIwU8dIfY8led
IhxSizEtJu29XeTvLEZIDBIhDEnxXOaVXhU8NAjfKJcG7ZInMRc6BnKE2Jz4ClyD
UOOvtwTyX27ZSTdCRCneQNxYt8WnZOGN/dlrIPP2UPfC12qZv85FtDkCgYEAwZiv
clSVu/FOB8WuZcu8xaZIYosvAjXAbRFsvRjMNf7swtGworslcjV5hgV+/JgIyooq
TMT6VefyY8475AFGfiyrOx5RHSL5G6IcTo7CW+e2B+cl0NdmKPafXxuKGS8hXkN8

NdCpBw3VNk+gNJZiIBGhNrt5mFInIUXGN95Vw/cCgYB4NFEZgN+YM5z+F4FRhHrO
5PRHFXwZfASzEMQLMwtXHFYdpSZyuE2rwpfqhQsw7Mryt3rmlOSVAlicARO18oxV
8LXgpxLzt2bl/SkD8JzVeL6k8LvwWdM1S5PF9D/tP5UWuBEYpOoHzGZHv/Djszkg
r3ROEKOpMyB4vhJElVPsYQKBgCXydcFlTwxN4b58QxjO/IWJeHrIaWh3fMZkOfo+
2BhlZacyTvs2z7o5PR7GCQQeEOsSdgQiQCeClYJ2tEqW7Whh/TQepyuc5VNDTWUg
mnxFSOr3pfPvpLi2zrpnvYP+Nv4xIDGOOs2FKpvAs9ha+dTrX7xItybiQ4VEFfxx
ekI1AoGBAKaMJPWH52UpgAzgW3CotoFv25kAcTnL4vJ6QK5SPKxWPRyduGaMnPgv
GawZhiCGM921ui/Zq521Keohik5Ak2flLtgMnnuzV6W2Nc7nfVqUqD7ekKhCPFog
C6t2lfrwq1aq2iZ6pkO4jpk7dsbeuUDHLoQIzZB3HVRrVhXfu/Ag
-----END RSA PRIVATE KEY-----
Shrouded Keybag: pbeWithSHA1And3-KeyTripleDES-CBC, Iteration 2000
Bag Attributes: <No Attributes>
Key Attributes: <No Attributes>
-----BEGIN RSA PRIVATE KEY-----
MIIJKAIBAAKCAgEA6qo7cNWI2bkLfbqYlOSYzXD/XtFdQHGjuk+EBnEyenSj7cq7
6DRbxhALbajx3FIZoQVR+bqOM9XInIRvl9Y2kOqTWG4t18z+5O/Tg7VNk3M9yUdl
cRA2kjG2CjWLe3U4ab7wNqG91+cguFaWtqu8BirRJvF6UdxudTo+mvmQh9l7PXHA
YK/clZ9C6+F/bLglLBvtXw7frK4TCdLvSZOZCXqKOy2pk7D2FPKhENpc1kzkbcxU
K6bFCrEZ5ONHJoudsHa+z3PVonduswzaOcmemH9F7/crbF+fiJ2/+EpbcyE/YBqQ
siJw6jXKGAczujLq9MZ3jH/OSdxs8PYW7GpAKC2yj3RV3rODrHrq4M+z5Vs38Rd5
bYnKKkKVSvOFyW7VlHEtp/2G5YxD+5AtcwKE3OYQiGr7+cF8SR8uN1JIV1XQ15YD
OhH7SD2mU5H9VXZmE79h/Zxl8qs1n+lrkG8d4zMYFkb6aPiWk6ed5pXyrZkVfZfN
pnUnUPViw3cNGKHP+5S7a8d/MN1KIKF4AuCfow+Tnya3m6dE3tJNZkkZ45BltYbc
iOjTOcNeUNKe3e/M+gHTIXGHRgE/T5DF6McTGEXOXb8ccmfO77OTrtODHNVX/F+e
gmVVnjaFRtvfNENYCnHHOgjONYIe2aqmOuSVKWOn9cAgQ13Pic66gAgrUbcCAwEA
AQKCAgAafy5HB1VOgBhOEeW47t3yFJhBNJK6/Bnop6HYuF9ixFjeOM4O4hqauqUP
tMvafMq4C8m9OPrNj44fXjTbiCyRIGuSyJ4Uh+kgIJRZgtdHlvObDpy9r3758vMD
JBijOBCWoCCvioPNJSPkrKJH7gxFBgCnzQxqweEXC4SWKyfOrxfjUDzQrgrSOjFs
QQIKXKCV4nFjEmLfNKihq1epkDuLfnK+daF4M7sVkm/tXzqsV/JTE4dNhn8pPeYW
mjMmfw1HJ6FT+1hcB/BP91uMut9ViMlO1wvoKMPywROOA/FfhpmJWHaOJqVpTLmN
DQMiOy2chexpyOSaInGKo77i4vxrVZd/yWzO8/AtYes9DA3Yd54OgTM+VQDY38+s
u6TejqOxKFXFKowSxPo3LH8pBOqO7Ve4IljrVZNsY/2z2Op9B694zX1+AVkuFkhR
MPzWQL+tLjbk7l8xnCG4vv9nupPFiCq+8ywigUkQGb7u+eFEvj7y/OROMxlIyzL+
F86efEWm2VAzb1vvLEloVWnnPXyHMVGqTfAXjdLiZ7OrgFQSOzd8vcxKKOJiDXMF
16m2hlfzSUlTqUSBgK/NuqjmOaTvOUwEOLIuCk73sdoS32wwe9CcNsr/D8AAhA6v
UPwuXhz6isw4DFyr6sbk5Z1zfuEi/omvtNAD4NCNUwqv/oMdwQKCAQEA/4B3v2Dd
sjlHCmXOaIZQdtritOwQF2IPvM5YzpOK7AOmw+dqczwZrX2CWYFQdCduJd8aOs6W
ohYTdLfp3klu+lNwTcMUdQ8dGYNRplZOCqzB+VlsJxjjGzVi18ih4LfVydDdmCGF
4JckABVlwkZ4trHhVOJCRQ+eskXYhck2xmskjnNQXStT14Fxlscnj9RFSBNTpuFt
WuPacpDm1pgRotwQ+UE6BLEdgTHPJPgxHbB7IenSc6+hCgUQPw2Ue9ovqX1y3VYP
wOTzLGLwTBKj1h33FWzpRdD1Qou+bibM8yaOikTzxN+CZ9rQKYbPK+AnXxWWJ8HK
91UUQG4WbEUKbQKCAQEA6x9dJA3mK8E3s8AOjYwTdSc5Bq18KIp+r4xkpNe83b5I
OLyAgR3HIQZCvK4ogPT4vXAP+qqPu4h1ne+G+/DILatUA6yDwBlMpbbQVf6jgsQv
QH7pSOCSoVJEPVvZdEMtYPdE7qvbKDP41B/dPEc87VTmSkFj7oThbrWkjKOy/C9R
Ywcuhtr4sMys3lUW1uQcOKgFi6d+6BvR6i5pKx5gNJrNm2iprtc+FtMAEDEdvEzf

```
JKoJb5VnWJnIgV6b4sFMlNwFAsOpWLIDJFO6o8qPagRUL6cEaOQhwVscgT/CkEjW
U8QVmOHER9VYgDYTkCPsM9NVaDkVkcahtm2EsIx2MwKCAQAOjZAT5TCOf+OE1/vk
s/vI9qy62fNVW+4wv6COnNRuRzH+CZNzsGD3tmEhV9/chF+7mnBB/Awvuqq1zz5j
YmdZReBLnlyO11T88m3Q8OddartbNnFweUX9SrXR/IqPVkC7CiMKL5sB3xSgj8Ym
d6hMq7nV4OMQoW3a7VLj8cc6eL9ELaPSBh9wrEdY5A3XH6pAcPk7OhJUJ5m/Bt7O
NMMr5Gf5XNk6LdQLxtr8l7BCkxJAn3+SHAXbsb7tuTVZZD95dTwzGoiS/ISoFJGo
lAzu9ark1UoF+jz4OxYoQgLV4Jgx+IzLac3A3tkCMaBbPffbLu17r/i3duB7gClM
i8bhAoIBAAspPvuiKUXOcxd2S3qOprHN/dgCTj2CbYcZzAFDOHTyPnSiLhty9Wvs
5ygbwFJ2zoP3mNqWVCJb1PF9WFQHVwlL874E53AsAOZuii862jov4+pU2/AYsPO1
cYXUIVtygOnOcvI9eDbNWMeBB7h/zTWa7R+jQQImjqj5uGjwsGNmP5/KzHucU2fZ
H2+ibGYhB/kjsLezaMPN5kYOk/F3DKwSmSYCbuOc1haxR3R9UGMK5yDPbYk9d11u
V2VEv3OhJ2h9lUNPVG+ga6PCOb5KtEodEH3+mKf+BFgwYgcEptDAlGP/j+Lk/LNI
Pl4/sugJvIGtS4WaCBsWacDGLCFRWZ8CggEBAP1u5rnL3aRXiizzdeuhy+wiqQE1
3t6YIVR2Wu3Fk9naRS96uqTUdqOD1Bi7SOYnf1Rg43erkNUThFTefiFfTvbL245X
WLCuBLSDH16bRvbO9Tfnj2artxwpiTQs8fRHTg3WOODz8roGs8jlQnn/m9bA144i
zhyDroy4xVOpJBgLguUOeiJoxEBi9MXw2fXx7xT8ITpzy/v9LO5BdBta+6CLNQ64
nyKE5+wS1fs6SRYc+X9zPwmOju6UrNHD8gsbzRPMep+s4IwCJ8clvV4z9kBkGYgB
86HBQ9Fgee+TkzDEj+ojwaZAyrH1G5oTmfaP2AlvHk8gBTPsOJuZjN2AXfE=
-----END RSA PRIVATE KEY-----

# efskeymgr -D 3661bf34:530116eb:1861a3eb:0cf71b91:91ca25e9


# efskeymgr -v
 Keystore content:
      Keystore owner ............ : uid 0
      Keystore mode ............. : admin: managed by EFS administrator
      Password changed last on .. : 04/18/07 at 04:08:43
     Private key:
           Algorithm : RSA_4096
          Fingerprint : f961c000:8305d1ba:65a72c48:eeafde52:b954bf78
            Validity : This key is valid.
     Private key:
           Algorithm : RSA_1024
          Fingerprint : 47dab395:99d8aa70:82659beb:700a4a6e:e232c8e4
             Validity : This key was deprecated on 04/20/07 at 07:35:45.
     Access key to keystore group/security
     Access key to keystore admin/
#
# efskeymgr -e keyfile4
Enter password for the new PKCS#12-protected file:
Enter the same password again:
#

OpenSSL> pkcs12 -in keyfile4 -info -nodes
```

```
Enter Import Password:
MAC Iteration 2000
MAC verified OK
PKCS7 Data
Certificate bag
Bag Attributes: <No Attributes>
subject=/CN=CLiC v4.0 17162B54
issuer=/CN=CLiC v4.0 17162B54
-----BEGIN CERTIFICATE-----
MIIErzCCApcCBBcWK1QwCwYJKoZIhvcNAQEBMBOxGzAZBgNVBAMTEkNMaUMgdjQu
MCAxNzE2MkI1NDAeFw0wNjA1MDgwMDAwMDBaFw0xNjA1MDgwMDAwMDBaMBOxGzAZ
BgNVBAMTEkNMaUMgdjQuMCAxNzE2MkI1NDCCAiIwDQYJKoZIhvcNAQEBBQADggIP
ADCCAgoCggIBAOqqO3DViNm5C326mJdEmM1w/17RXUBxo7pPhAZxMnpOo+3Ku+gO
W8YQC22o8dxSGaEFUfm6jjPVyJyEb5fWNpDqk1huLdfM/uTv04O1TZNzPclHZXEQ
NpIxtgo1i3t1OGm+8DahvdfnILhWlrarvAYqOSbxelHcbnU6Ppr5kIfZez1xwGCv
3JWfQuvhf2y4JSwb7V8O36yuEwnS7OmdGQl6itMtqZOw9hTyoRDaXNZM5G3MVCum
xQqxGeTjRyaLnbB2vs9z1aJ3brMM2jnJnph/Re/3K2xfn4idv/hKW3MhP2AakLIi
cOo1yhgHM7oy6vTGd4x/zkncbPD2FuxqQCgtso9OVd69A6x66uDPs+VbN/EXeW2J
yipClUr9Bclu1ZRxLaf9huWMQ/uQLXMChN9GEIhq+/nBfEkfLjdSSFdVONeWAzoR
+0g9plOR/VV2ZhO/Yf2cZfKrNZ/pa5BvHeMzGBZG+mj4lpOnneaV8q2ZFX2XzaZ1
J1D1YsN3DRihz/uUu2vHfzDdSiCheALgn6MPk58mt5unRN7STWZJGeOQZbWG3Ijo
O9HDXlDSnt3vzPoBOyFxhOYBPO+QxejHExhF9F2/HHJnzu+zk67dAxzVV/xfnoJl
VZ42hUbb3zRDWApxx9II9DWCHtmqpjrklSltJ/XAIENdz4nOuoAIK1G3AgMBAAEw
DQYJKoZIhvcNAQEFBQADggIBALr+B3OWQPl8cGvP/ZPYV7KLyKH+TCQwRPp/WOye
wbDDrhwcVsEKeYhtFDcifVsXdM4FGbC14Fbv7UEQpWOhQO1IHHUOutXEyFe1O7wh
SItLBJJCKjvHPpwyDmey3pe/JsdKWt3hWWD8e7TwjcBY9lp7Z3fWH11CTnI8VA+3
12mOY6DsBTwa1UfFoYR4Fl2GBzYvN8j3wXpdPGznvd4s/VHcopcu/ya6aEU+ZMOq
ehAnlfEDw9ASVHYzt77+XxskGicRV+OTe5X/pt4ja89MjRY/nXFO46cnxTivH2eI
xI5Z47yFdyGVcsfWkhdIyV69XZ+WN29G7z143QyUWA1FFFq9vjF4KPERou5nO3HU
floFhRamhJDKyuZBHCcC4fw1CcXUK19vRUps1i1wM/oC9Iwd4h8oj+PFN/+tR53u
rbdYsMAP71duHon2AWTeaZlgRDx9Ts9vqywOoO5C/OXXEnfKuF+AfDuv3jxHh84M
jRkEBOLasqZWrEDIMUoXyi7ypICQ5NOhroOPhw7QQjaJb7CeDtCzCtOwosqQODdL
87OmEWkNLY2pvg8T6VgqepNpqNIrKw4y6ILViyqPLT6/lnri7c7p5qm8XKgkyMdl
rPpS/9ZtxfujJSjjIAQrNKjoHhewrBBOgq2JZ37PU+BB/TOmjWOdODX4q5QJOcXG
TUk7
-----END CERTIFICATE-----
Shrouded Keybag: pbeWithSHA1And3-KeyTripleDES-CBC, Iteration 2000
Bag Attributes: <No Attributes>
Key Attributes: <No Attributes>
-----BEGIN RSA PRIVATE KEY-----
MIICXQIBAAKBgQC8x7LSx3WGHxh2gTGEBOVJHOpA/rXM5AZCCnZzlpT3s3fSvEKq
nABybotOTjLM6X1axBZGDzImqTMTcnObbnjtqra83FG6j/OAzlhmcrUkCdFnJk7Q
OYOihV+xvvzZD2Mv2fyy5OXMSkhD2Nvrz6SfuZpcIYwcUiYSdmO/dga3SwIDAQAB
AoGAHJ7KtyxYiSQiEXv7wE/9B616lJ5TOgOItAQhujDMVpNlvqlzvVRCYxbsWzks
eCbhWNydTd4plHVBfO4gwS2/SnHGwnQwO6LcVgkqONSDqbf2yTJGBUDUwmZOqOxp
```

126
AIX V6 Advanced Security Features: Introduction and Configuration

HPrFsl99VzrJM+/pgEQcqEbdwc7l82NgCBcuPYtNpNSKL9ECQQD6X5os5wTbWdzo
ukQlWTmKWpdn3K+muDwIwy2oGesSY2G24i5IZcrrsX5gzUlSI4Z22zG9m1JnaItC
/b44Tl9bAkEAwQW/8OG4Vd216ULbLNoasoA1GVGfIiVJCzqzEO2nt19bQwwNwwrt
zmzJVczCi8cEWSFQxbe/3geOAYiNfoT6OQJBAO+1/IylGndQODpZq8j+ZiLanT8p
gVrj5UaWZ+4b8n6GfBV288OI+IEOTMUthteHf8PoFPVt8jVjWIHpIugROSUCQBUp
XTa6eGwph3UQWFkdfEnPloK1GR4OOkLZ56HWfEm1UZsTKjsU6qdz88rNTRLn+ckP
xvw2PfnImEAAyYpyZ9ECQQDSx4/ZF2/Uk/lOBe5SOBkYKKDnVrioXBjLqn161ERa
8ABuiz3EeI5knBx/sd8FVhNF+Izka5qcA4rd7XYvar1s
-----END RSA PRIVATE KEY-----
Shrouded Keybag: pbeWithSHA1And3-KeyTripleDES-CBC, Iteration 2000
Bag Attributes: <No Attributes>
Key Attributes: <No Attributes>
-----BEGIN RSA PRIVATE KEY-----
MIIJKAIBAAKCAgEA6qo7cNWI2bkLfbqYlOSYzXD/XtFdQHGjuk+EBnEyenSj7cq7
6DRbxhALbajx3FIZoQVR+bqOM9XInIRvl9Y2kOqTWG4t18z+5O/Tg7VNk3M9yUdl
cRA2kjG2CjWLe3U4ab7wNqG91+cguFaWtqu8BirRJvF6UdxudTo+mvmQh9l7PXHA
YK/clZ9C6+F/bLglLBvtXw7frK4TCdLvSZOZCXqKOy2pk7D2FPKhENpc1kzkbcxU
K6bFCrEZ5ONHJoudsHa+z3PVonduswzaOcmemH9F7/crbF+fiJ2/+EpbcyE/YBqQ
siJw6jXKGAczujLq9MZ3jH/OSdxs8PYW7GpAKC2yj3RV3r0DrHrq4M+z5Vs38Rd5
bYnKKkKVSvOFyW7VlHEtp/2G5YxD+5AtcwKE3OYQiGr7+cF8SR8uN1JIV1XQ15YD
OhH7SD2mU5H9VXZmE79h/Zxl8qs1n+lrkG8d4zMYFkb6aPiWk6ed5pXyrZkVfZfN
pnUnUPViw3cNGKHP+5S7a8d/MN1KIKF4AuCfow+Tnya3m6dE3tJNZkkZ45BltYbc
iOjTOcNeUNKe3e/M+gHTIXGHRgE/T5DF6McTGEXOXb8ccmfO77OTrtODHNVX/F+e
gmVVnjaFRtvfNENYCnHHOgjONYIe2aqmOuSVKWOn9cAgQl3Pic66gAgrUbcCAwEA
AQKCAgAafy5HB1VOgBhOEeW47t3yFJhBNJK6/Bnop6HYuF9ixFjeOM4O4hqauqUP
tMvafMq4C8m9OPrNj44fXjTbiCyRIGuSyJ4Uh+kgIJRZgtdHlvObDpy9r3758vMD
JBijOBCWoCCvioPNJSPkrKJH7gxFBgCnzQxqweEXC4SWKyfOrxfjUDzQrgrSOjFs
QQIKXKCV4nFjEmLfNKihq1epkDuLfnK+daF4M7sVkm/tXzqsV/JTE4dNhn8pPeYW
mjMmfw1HJ6FT+1hcB/BP91uMut9ViMlO1wvoKMPywROOA/FfhpmJWHaOJqVpTLmN
DQMiOy2chexpyOSaInGKo77i4vxrVZd/yWzO8/AtYes9DA3Yd54OgTM+VQDY38+s
u6TejqOxKFXFKowSxPo3LH8pBOqO7Ve4IljrVZNsY/2z2Op9B694zX1+AVkuFkhR
MPzWQL+tLjbk7l8xnCG4vv9nupPFiCq+8ywigUkQGb7u+eFEvj7y/OROMxlIyzL+
F86efEWm2VAzb1vvLEloVWnnPXyHMVgqTfAXjdLiZ7OrgFQSOzd8vcxKKOJiDXMF
16m2hlfzSUlTqUSBgK/NuqjmOaTvOUwEOLIuCk73sdoS32wwe9CcNsr/D8AAhA6v
UPwuXhz6isw4DFyr6sbk5Z1zfuEi/omvtNAD4NCNUwqv/oMdwQKCAQEA/4B3v2Dd
sjlHCmXOaIZQdtritOwQF2IPvM5YzpOK7AOmw+dqczwZrX2CWYFQdCduJd8aOs6W
ohYTdLfp3klu+lNwTcMUdQ8dGYNRplZOCqzB+VlsJxjjGzVi18ih4LfVydDdmCGF
4JckABVlwkZ4trHhVOJCRQ+eskXYhck2xmskjnNQXStT14Fxlscnj9RFSBNTpuFt
WuPacpDm1pgRotwQ+UE6BLEdgTHPJPgxHbB7IenSc6+hCgUQPw2Ue9ovqX1y3VYP
wOTzLGLwTBKj1h33FWzpRdD1Qou+bibM8yaOikTzxN+CZ9rQKYbPK+AnXxWWJ8HK
91UUQG4WbEUKbQKCAQEA6x9dJA3mK8E3s8AOjYwTdSc5Bq18KIp+r4xkpNe83b5I
OLyAgR3HIQZCvK4ogPT4vXAP+qqPu4h1ne+G+/DILatUA6yDwBlMpbbQVf6jgsQv
QH7pSOCSoVJEPVvZdEMtYPdE7qvbKDP41B/dPEc87VTmSkFj7oThbrWkjKOy/C9R
Ywcuhtr4sMys3lUW1uQcOKgFi6d+6BvR6i5pKx5gNJrNm2iprtc+FtMAEDEdvEzf
JKoJb5VnWJnIgV6b4sFMlNwFAsOpWLIDJFO6o8qPagRUL6cEaOQhwVscgT/CkEjW

```
U8QVmOHER9VYgDYTkCPsM9NVaDkVkcahtm2EsIx2MwKCAQAOjZAT5TCOf+OE1/vk
s/vI9qy62fNVW+4wv6COnNRuRzH+CZNzsGD3tmEhV9/chF+7mnBB/Awvuqq1zz5j
YmdZReBLnlyO11T88m3Q8OddartbNnFweUX9SrXR/IqPVkC7CiMKL5sB3xSgj8Ym
d6hMq7nV4OMQoW3a7VLj8cc6eL9ELaPSBh9wrEdY5A3XH6pAcPk7OhJUUJ5m/Bt7O
NMMr5Gf5XNk6LdQLxtr817BCkxJAn3+SHAXbsb7tuTVZZD95dTwzGoiS/ISoFJGo
lAzu9ark1UoF+jz4OxYoQgLV4Jgx+IzLac3A3tkCMaBbPffbLu17r/i3duB7gClM
i8bhAoIBAAspPvuiKUXOcxd2S3qOprHN/dgCTj2CbYcZzAFDOHTyPnSiLhty9Wvs
5ygbwFJ2zoP3mNqWVCJb1PF9WFQHVwlL874E53AsAOZuii862jov4+pU2/AYsPO1
cYXUIVtygOnOcvI9eDbNWMeBB7h/zTWa7R+jQQImjqj5uGjwsGNmP5/KzHucU2fZ
H2+ibGYhB/kjsLezaMPN5kYOk/F3DKwSmSYCbuOc1haxR3R9UGMK5yDPbYk9d11u
V2VEv3OhJ2h9lUNPVG+ga6PCOb5KtEodEH3+mKf+BFgwYgcEptDAlGP/j+Lk/LNI
Pl4/sugJvIGtS4WaCBsWacDGLCFRWZ8CggEBAP1u5rnL3aRXiizzdeuhy+wiqQE1
3t6YIVR2Wu3Fk9naRS96uqTUdqOD1Bi7SOYnf1Rg43erkNUThFTefiFfTvbL245X
WLCuBLSDH16bRvbO9Tfnj2artxwpiTQs8fRHTg3WOODz8roGs8jlQnn/m9bA144i
zhyDroy4xVOpJBgLguUOeiJoxEBi9MXw2fXx7xT8ITpzy/v9LO5BdBta+6CLNQ64
nyKE5+wS1fs6SRYc+X9zPwmOju6UrNHD8gsbzRPMep+s4IwCJ8clvV4z9kBkGYgB
86HBQ9Fgee+TkzDEj+ojwaZAyrH1G5oTmfaP2AlvHk8gBTPsOJuZjN2AXfE=
-----END RSA PRIVATE KEY-----
```

2.5.13 User public key

Finding a user public key is also important, especially when you want to integrate user keys in a certificate infrastructure.

In Example 2-35 on page 129, we offer an example of how to discover the value of the public key as follows:

► Export the current content of the keystore into keyfile using the `efskeymgr -e keyfile` command.

► Export the content of keyfile using the command `openssl pkcs12 -in keyfile -nodes -out file1`.

► Display the content of file1 using the `cat file1` command.

► Edit file1 to remove the private bag and leave only the content of the public bag. The public key is enclosed in a certificate. We displayed the content of file1 after it has been edited.

► Display the public key using the `openssl x509 -in file1 -pubkey -noout` command.

Example 2-35   Displaying the RSA public key

```
# efskeymgr -e keyfile
Enter password for the new PKCS#12-protected file:
Enter the same password again:
# openssl pkcs12 -in keyfile -nodes -out file1
# cat file1
Bag Attributes: <No Attributes>
subject=/CN=CLiC v4.0 7CB79796
issuer=/CN=CLiC v4.0 7CB79796
-----BEGIN CERTIFICATE-----
MIICrzCCAZcCBHy3l5YwCwYJKoZIhvcNAQEBMBOxGzAZBgNVBAMTEkNMaUMgdjQu
MCA3QOI3OTc5NjAeFwOwNjA1MDgwMDAwMDBaFwOxNjA1MDgwMDAwMDBaMBOxGzAZ
BgNVBAMTEkNMaUMgdjQuMCA3QOI3OTc5NjCCASIwDQYJKoZIhvcNAQEBBQADggEP
ADCCAQoCggEBAMtIfI6Pc8UTNJj5GfAHBUCej7ftwsBXDJEheHsioHydNjnbeDfU
K8JLWOK8j7Sxy/WZx/DQrwMrfHBgrIlU9kNu7PY7rjHvi1Tv58EZpNZqQXTWo4ag
OdGFQ8GOvISUhWzRtL5+tclex66ULDEStBnQBmAuwOQUBdIUtpnvKlVJsphaRlc9
wKW8KDe2yiOJ2ueSvspIEtW8PbQGuVPckrgu8HzfUAoybX654aoEKOQc0+ZaF12S
dZiO+5Zls2dGbdVjoItLLGe8y4NWQvLfe9oyBygz6NVaDUaVGIe/MpCZjvRboyMI
PXG4ghwsY75KNc2NOMafWRgHdXpcDNCh7cECAwEAATANBgkqhkiG9wOBAQUFAAOC
AQEAxub8XKecKI4iVscGLOtQiP91kXgn6Pfd4gcBWqLVznsTW4K+h2Jy8Zocjv1e
Ypgf5Zny9+GwOOJ5C8FMOqfTmWVNiqbEhj4vudzfhA+i8nObKsiLA8KvyQUtsmy+
IyTp3YYz3mvipKSSArVYOEh9Hibtl3J6GO1am2KlHPDKKwfNkLtOvDkWihjtODop
IYxJBFCr3LzcZP1GnvcvfDbQdGqHuxwjAiHwwyph4QIixG4irLhx4QtFMUfEthAf
x/U6fW2h7uVy4E46Ra+/965zOeeLBJgSZaRiu9fRAcxp5U4a4Q6LTlvwcbSqAiQV
9CNCySOPidTe6gHrmqzC5ZvQsA==
-----END CERTIFICATE-----
Bag Attributes: <No Attributes>
Key Attributes: <No Attributes>
-----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEAyOh8jo9zxRMOmPkZ8AcFQJ6Pt+3CwFcMkSF4eyKgfJO2Odt4
N9QrwktbQryPtLHL9ZnH8NCvAyt8cGCsiVT2Q27s9juuMe+LVO/nwRmk1mpBdNaj
hqA50YVDwY68hJSFbNGOvn61yV7HrpQsMRKOGdAGYC7DRBQFOhS2me8qVUmymFpG
Vz3ApbwoN7bKLQna55K+ykgS1bw9tAa5U9ySuC7wfN9QCjJtfrnhqgQo5BzT5loX
XZJ1mI77lmWzZOZt1WOgiOssZ7zLg1ZC8t972jIHKDPo1VoNRpUYh78ykJmO9Fuj
Iwg9cbiCHCxjvko1zYO4xp9ZGAd1elwMOKHtwQIDAQABAoIBAC/rE9VlElSjzNO8
cHVIql/LoLQqGyzMuDOjxys7g9osEuK7jdCXNoNzOPzfxv5ApQPtbpihEIguQYSZ
/UHYIb7OkJQQiGTkAxOQA7DckyMp4+kfSWfMMCToBYtEUsLr4bG/kxwOcVqY+Xf5
HJuqvYKSK1aT31qWX7LgSBTsMC3vKFL1A1KaelIkf9Oez2a/+D15dAfWgbwZBvBR
KJXOkX08l3FEn18EkrJ9AUavzoM32kWyvhwoPXICikm8w4n85xLsnTmWC3cYiZdu
RDSM6BHIYsBBmMDRUPKNzXf/9freGFcNEVThzavLb7YqXqqvbd8EufRv12P4ibrg
pwyGgwsCgYEA6WsTgdJdbaE4GnOE9LdC9O8UjrfgCADl67mAicIML3UY9hO1tmRg
+QK4YOZRG8s337UNdGjrBAh5Mh42yu1WWgNK1c1HVNPmLMqNPXM6UGvQn98bf+3s
ON+KsJ14iN1rsgCF8tLvZo7WUu49K4m+Od1N1h5tyHJYC5G95qdHCtsCgYEA3vMS
pYmLW1WaBPcQ4bSGnivujpKTXdFzy64UZKGP+HsFQ1SmXdLLFo7RsNN1DhdSg7K+
```

Chapter 2. Encrypted File System   **129**

```
tR6Oiy+f3InOSyMSlf2AgBGODQzmLWAbXIiVYBU+q3DCOcaOq4GVY1phm6RhPRZ6
cx/lwgFD2LNOmHVFtGOmvHY8H6OeWr9XRFwNtpMCgYAHIHbTAupJW5/L5vULR8Eq
reZxyRl1BRoADYToL92RYWDmutOWAxCW6cdZZg7Z62WEVhHYGQ/76opQxm1j2sVF
ME6Lop3n8CRZ/8E6PBdutywChZxgVpJhekHQzK4Od7w/DhcI5/nZRskydQW3G5iM
Sb6DqCAv+5XGnKeQsFDR+wKBgQDJsqpZ7FeQ556Rg1tYdlObYS9O+LgtlkHmMLTu
XWO8br9SLJDeLMfivo5iJMuvVdYmo1z4yo42yXOcredd/nrgCxlnw5xaeiL7Rgk6
664H4PBzdW4rDlOBHZii66+GeW4nL+DTqjXYEADrWV7QVrgbBOyYb3bxSVM+OgAH
LiiR1QKBgQCjyvFW1OzSwPcsyb1AaktiqGBqD6Ne/XbSw69BiScwHJVyhOvmXAx+
QTEQca8uVbWvxCF6r542CI1pT8mwzl3twQ3OuiaUmnEgXhYtGdQ8MPNYg6EO3Ncp
czd1J5qwVKDZf6BMkOByAysrG42c5xDg/SpaYSj/fOZOS9wfNERVGA==
-----END RSA PRIVATE KEY-----
# cat file1
subject=/CN=CLiC v4.0 7CB79796
issuer=/CN=CLiC v4.0 7CB79796
-----BEGIN CERTIFICATE-----
MIICrzCCAZcCBHy3l5YwCwYJKoZIhvcNAQEBMBOxGzAZBgNVBAMTEkNMaUMgdjQu
MCA3Q0I3OTc5NjAeFwOwNjA1MDgwMDAwMDBaFwOxNjA1MDgwMDAwMDBaMBOxGzAZ
BgNVBAMTEkNMaUMgdjQuMCA3Q0I3OTc5NjCCASIwDQYJKoZIhvcNAQEBBQADggEP
ADCCAQoCggEBAMtIfI6Pc8UTNJj5GfAHBUCej7ftwsBXDJEheHsioHydNjnbeDfU
K8JLWOK8j7Sxy/WZx/DQrwMrfHBgrIlU9kNu7PY7rjHvi1Tv58EZpNZqQXTWo4ag
OdGFQ8GOvISUhWzRtL5+tclex66ULDEStBnQBmAuwOQUBdIUtpnvKlVJsphaRlc9
wKW8KDe2yiOJ2ueSvspIEtW8PbQGuVPckrgu8HzfUAoybX654aoEKOQcO+ZaF12S
dZiO+5Zls2dGbdVjoItLLGe8y4NWQvLfe9oyBygz6NVaDUaVGIe/MpCZjvRboyMI
PXG4ghwsY75KNc2NOMafWRgHdXpcDNCh7cECAwEAATANBgkqhkiG9w0BAQUFAAOC
AQEAxub8XKecKI4iVscGLOtQiP91kXgn6Pfd4gcBWqLVznsTW4K+h2Jy8Zocjv1e
Ypgf5Zny9+GwOOJ5C8FMOqfTmWVNiqbEhj4vudzfhA+i8nObKsiLA8KvyQUtsmy+
IyTp3YYz3mvipKSSArVYOEh9Hibtl3J6GO1am2KlHPDKKwfNkLtOvDkWihjtODop
IYxJBFCr3LzcZP1GnvcvfDbQdGqHuxwjAiHwwyph4QIixG4irLhx4QtFMUfEthAf
x/U6fW2h7uVy4E46Ra+/965zOeeLBJgSZaRiu9fRAcxp5U4a4Q6LTlvwcbSqAiQV
9CNCySOPidTe6gHrmqzC5ZvQsA==
-----END CERTIFICATE-----
# openssl x509 -in file1 -pubkey -noout
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAyOh8jo9zxRMOmPkZ8AcF
QJ6Pt+3CwFcMkSF4eyKgfJO2Odt4N9QrwktbQryPtLHL9ZnH8NCvAyt8cGCsiVT2
Q27s9juuMe+LVO/nwRmk1mpBdNajhqA5OYVDwY68hJSFbNGOvn61yV7HrpQsMRKO
GdAGYC7DRBQFOhS2me8qVUmymFpGVz3ApbwoN7bKLQna55K+ykgS1bw9tAa5U9yS
uC7wfN9QCjJtfrnhqgQo5BzT5loXXZJ1mI77lmWzZOZt1WOgiOssZ7zLg1ZC8t97
2jIHKDPo1VoNRpUYh78ykJmO9FujIwg9cbiCHCxjvko1zYO4xp9ZGAd1elwMOKHt
wQIDAQAB
-----END PUBLIC KEY-----
```

2.5.14 Importance of deprecated keys

Private keys are vital to get file access. They are used to decrypt the file symmetric key that was encrypted with the user public key at file creation. In this way, the user obtains the symmetric encryption key and can get access to data in the file.

The scenario in Example 2-36 shows this situation as follows:

► The `efsmgr -l` command displays the key that can have access to the file.

► The key that can access the file is present in the user keystore and is valid as shown by the `efskeymgr -v` command.

► The user changes his or her private key using the `efskeymgr -R` command.

► The newly-generated key is stored in the user keystore along with the old key. The old key is marked as deprecated.

► Although the old key is deprecated, it can be used to gain access to the file.

► The `efskeymgr -V` command shows that the old key is still loaded in the current shell and the new key has not been automatically loaded.

► The user deletes the deprecated key using the `efskeymgr -D` command and the `efskeymgr -v` command confirms that the old key has been deleted from the keystore.

► The `efskeymgr -o ksh` command loads the content of the keystore.

► The `efskeymgr -V` command shows that only the new key is available. The old key has been deleted.

► Since it does not have the proper key, the user is no longer able to access the file.

Example 2-36   Using deprecated keys to access old files

```
$ efsmgr -l file
EFS File information:
 Algorithm: AES_128_CBC
List of keys that can open the file:
 Key #1:
  Algorithm      : RSA_1024
  Who            : uid 205
  Key fingerprint : 30412121:e5a7b90d:dba7dd19:2c45b1e0:c331c09f
$ efskeymgr -v
 Keystore content:
     Keystore owner ............ : uid 205
     Keystore mode ............. : admin: managed by EFS administrator
     Password changed last on .. : 05/10/07 at 09:41:20
```

```
                Private key:
                     Algorithm : RSA_1024
                    Fingerprint : 30412121:e5a7b90d:dba7dd19:2c45b1e0:c331c09f
                      Validity : This key is valid.
        $ efskeymgr -R RSA_1024
        $ efskeymgr -v
        user3's EFS password:
         Keystore content:
               Keystore owner ............ : uid 205
               Keystore mode ............. : admin: managed by EFS administrator
               Password changed last on .. : 05/10/07 at 09:41:20
              Private key:
                   Algorithm : RSA_1024
                  Fingerprint : 44b0c7e1:53384985:ca1be27e:864b9431:3a57c0d1
                    Validity : This key is valid.
              Private key:
                   Algorithm : RSA_1024
                  Fingerprint : 30412121:e5a7b90d:dba7dd19:2c45b1e0:c331c09f
                    Validity : This key was deprecated on 05/10/07 at 19:24:35.
        $ cat file
        data in encrypted file
        $ efskeymgr -V
        List of keys loaded in the current process:
         Key #0:
           Kind .................... User key
           Id   (uid / gid) ........ 205
           Type ................... Private key
           Algorithm ............... RSA_1024
           Validity ................ Key is valid
           Fingerprint ..............
        30412121:e5a7b90d:dba7dd19:2c45b1e0:c331c09f
        $ efskeymgr -D 30412121:e5a7b90d:dba7dd19:2c45b1e0:c331c09f
        user3's EFS password:
        $ efskeymgr -v
         Keystore content:
               Keystore owner ............ : uid 205
               Keystore mode ............. : admin: managed by EFS administrator
               Password changed last on .. : 05/10/07 at 09:41:20
              Private key:
                   Algorithm : RSA_1024
                  Fingerprint : 44b0c7e1:53384985:ca1be27e:864b9431:3a57c0d1
                    Validity : This key is valid.

        $ efskeymgr -o ksh
        user3's EFS password:
```

```
$ k
$ efskeymgr -V
List of keys loaded in the current process:
 Key #0:
   Kind .................... User key
   Id   (uid / gid) ........ 205
   Type ................... Private key
   Algorithm ............... RSA_1024
   Validity ................ Key is valid
   Fingerprint ...........
44b0c7e1:53384985:ca1be27e:864b9431:3a57c0d1
$ cat file
cat: 0652-050 Cannot open file.
```

2.6 Group management

In this section, we show how group management functions are integrated with new cryptographic features.

2.6.1 Defining groups

Every time you create a new group, you must define its security related information. Group security attributes that have been added to /etc/security/group are shown in Example 2-37.

Example 2-37   New attributes added in group security configuration file

```
# cat /etc/security/group|grep -ip group2
group2:
        admin = false
        adms = root
    efs_keystore_access = file
    efs_initialks_mode = admin
    efs_keystore_algo = RSA_1024
```

In order to support the new features that have been added for group security, three fields have been also added to SMIT screens used for group management, as shown in Example 2-38.

Example 2-38   New fields added in the SMIT screen for group creation

```
Add a Group

Type or select values in entry fields.
Press Enter AFTER making all desired changes.


                                                    [Entry Fields]
* Group NAME                                       [group1]
  ADMINISTRATIVE group?                   false
+
  Group ID                      []
#
  USER list                     []
+
  ADMINISTRATOR list               []
+
  Projects                     []
+
  Initial Keystore Mode             []
+
  Keystore Encryption Algorithm        []
+
  Keystore Access                []
+


F1=Help               F2=Refresh              F3=Cancel
F4=List
Esc+5=Reset           Esc+6=Command           Esc+7=Edit
Esc+8=Image
Esc+9=Shell             Esc+0=Exit                Enter=Do
```

The meaning of the newly-added fields in Example 2-38 are:

Initial Keystore Mode

This field describes the initial mode of the user keystore. The value of this field can be either admin or guard, corresponding to either root admin mode or root guard admin mode.

Keystore Encryption Algorithm

This field describes the algorithm and the length of the keys used for user private and public keys. If you change the value of this field, the corresponding attribute from the user stanza from /etc/security/user is changed accordingly. The value can be one of the following:

- RSA_1024
- RSA_2048
- RSA_4096

Key store access   The value of this field must be file or none. File means local /var/efs/groups/*. None means no keystore. When the group is created by root, the security attributes from group stanza in /etc/security/group are initialized using the default values. However, at this time there is no keystore for the group.

When you define a group in root admin mode, root is automatically assigned as a group administrator.

2.6.2 Group keystore design and operations

Group keystore internal design is similar to user keystore design.

Operations that modify a group keystore include:

- ► Changing algorithms
- ► Changing key length
- ► Generating a new key
- ► Deleting a deprecated key

2.6.3 Defining a group and creating a group keystore

In Example 2-39 on page 136, we show how to create a group keystore for a group named group1 by doing the following:

- ► Define the group named group1. This group does not contain any users yet. In /var/efs/groups, a new directory named group1 is created. This directory does not contain any keystores yet.
- ► Create a user named user3. Add user3 to group1.
- ► Trying to create a group keystore at this time fails again because although the group does contain a member, the user does not have any keystores yet.

- The password for user3 is initialized so the user keystore is created.
- The key store for group1 is now successfully created.
- The access key to group1 keystore has been automatically sent to user3 keystore. When user3 logs in, the access key for group1 keystore will be automatically loaded.

Example 2-39   Creating a group keystore

```
# mkgroup group1
# lsgroup -a id users adms group1
group1 id=207 users=
# ls -al /var/efs/groups
total 0
drwx------    4 root     system          256 May 02 14:12 .
drwxr-xr-x    5 root     system          256 May 02 13:34 ..
-rw-------    1 root     system            0 May 02 13:34 .lock
drwx------    2 root     system          256 May 02 14:12 group1
drwx------    2 root     system          256 May 02 13:34 security
# ls -al /var/efs/groups/group1
total 0
drwx------    2 root     system          256 May 02 14:12 .
drwx------    4 root     system          256 May 02 14:12 ..
-rw-------    1 root     system            0 May 02 14:12 .lock
# efskeymgr -C group1
Encryption framework returned an error: (The group does not contain any
member with EFS crypto info)
kernel: The system call does not exist on this system.
# mkuser user3;chgroup users=user3 group1;lsgroup -a id users adms
group1
group1 id=207 users=user3
# efskeymgr -C group1
Encryption framework returned an error: (The group does not contain any
member with EFS crypto info)
kernel: The system call does not exist on this system.
# passwd user3
Changing password for "user3"
user3's New password:
Enter the new password again:
# efskeymgr -C group1
# ls -al /var/efs/groups/group1
total 8
drwx------    2 root     system          256 May 02 14:27 .
drwx------    4 root     system          256 May 02 14:12 ..
-rw-------    1 root     system            0 May 02 14:12 .lock
-rw-------    1 root     system         1914 May 02 14:27 keystore
```

```
$ id
uid=209(user3) gid=1(staff) groups=207(group1)
$ efskeymgr -v
user3's EFS password:
 Keystore content:
     Keystore owner ............ : uid 209
     Keystore mode ............. : admin: managed by EFS administrator
     Password changed last on .. : 05/02/07 at 14:27:21
   Private key:
         Algorithm : RSA_1024
       Fingerprint : 7dc53a23:b8d1d7f0:c6f43e88:ae7be372:f585f4e4
          Validity : This key is valid.
   Access key to keystore group/group1
```

2.6.4 Sending the group keystore access key to a user

When running in root admin mode, you can use the **efskeymgr** command to send
an access key for a group store to a user keystore using the following steps
shown in Example 2-40:

► user5 has no access key for group1.

► The group keystore access key is sent to the user keystore.

► user5 now has the group keystore access key loaded into his or her own user
  keystore. The key is not automatically loaded into active processes.

Example 2-40   Sending an group keystore access key to a user

```
$ id
uid=207(user5) gid=1(staff) groups=202(group1)
$ efskeymgr -v
 Keystore content:
     Keystore owner ............ : uid 207
     Keystore mode ............. : admin: managed by EFS administrator
     Password changed last on .. : 05/03/07 at 17:39:43
   Private key:
         Algorithm : RSA_1024
       Fingerprint : 49e42532:e8c471bf:2be9ba03:3f9d12b4:4ba01cea
          Validity : This key is valid.
# id
uid=0(root) gid=0(system)
groups=2(bin),3(sys),7(security),8(cron),10(audit),11(lp)
# efskeymgr -k group/group1 -s user/user5
$ id
uid=207(user5) gid=1(staff) groups=202(group1)
```

```
$ efskeymgr -v
 Keystore content:
      Keystore owner ............ : uid 207
      Keystore mode ............. : admin: managed by EFS administrator
      Password changed last on .. : 05/03/07 at 17:39:43
    Private key:
           Algorithm : RSA_1024
         Fingerprint : 49e42532:e8c471bf:2be9ba03:3f9d12b4:4ba01cea
            Validity : This key is valid.
    Access key to keystore group/group1
$ efskeymgr -V
List of keys loaded in the current process:
 Key #0:
    Kind .................... User key
    Id   (uid / gid) ........ 207
    Type .................... Private key
    Algorithm ............... RSA_1024
    Validity ................ Key is valid
    Fingerprint .............
49e42532:e8c471bf:2be9ba03:3f9d12b4:4ba01cea
```

Important: When you add a user to a group, either using `chuser` or `chgroup`, the group access key is automatically sent to the user keystore.

When running in root admin mode, root can add himself as a member of any group and will automatically receive the group access key. Any group member user automatically accepts the group access key in his or her keystore.

2.6.5 Removing the group keystore access key from a user keystore

When running in root admin mode, you can use the `efskeymgr` command to remove an access key for a group keystore from a user keystore using the following steps shown in Example 2-41 on page 139:

► user5 has the access key for group1. The group access key is loaded in the current shell.

► The group keystore access key is removed from the user keystore.

► Although user5 does not have the group keystore in his or her user keystore any longer, the group keystore access key is not automatically unloaded from active processes.

Example 2-41   Sending an group keystore access key to a user

```
$ id
uid=207(user5) gid=1(staff) groups=202(group1)
$ efskeymgr -v
 Keystore content:
      Keystore owner ............ : uid 207
      Keystore mode ............. : admin: managed by EFS administrator
      Password changed last on .. : 05/03/07 at 17:39:43
    Private key:
          Algorithm : RSA_1024
        Fingerprint : 49e42532:e8c471bf:2be9ba03:3f9d12b4:4ba01cea
           Validity : This key is valid.
    Access key to keystore group/group1
$ efskeymgr -v
 Keystore content:
      Keystore owner ............ : uid 207
      Keystore mode ............. : admin: managed by EFS administrator
      Password changed last on .. : 05/03/07 at 17:39:43
    Private key:
          Algorithm : RSA_1024
        Fingerprint : 49e42532:e8c471bf:2be9ba03:3f9d12b4:4ba01cea
           Validity : This key is valid.
    Access key to keystore group/group1
$ efskeymgr -V
List of keys loaded in the current process:
 Key #0:
   Kind .................... User key
   Id   (uid / gid) ........ 207
   Type .................... Private key
   Algorithm ............... RSA_1024
   Validity ................ Key is valid
   Fingerprint .............
49e42532:e8c471bf:2be9ba03:3f9d12b4:4ba01cea
 Key #1:
   Kind .................... Group key
   Id   (uid / gid) ........ 202
   Type .................... Private key
   Algorithm ............... RSA_1024
   Validity ................ Key is valid
   Fingerprint .............
5100b522:fbee1dbe:2aeddbd3:c2e51fea:86ab4b5c
# id
uid=0(root) gid=0(system)
groups=2(bin),3(sys),7(security),8(cron),10(audit),11(lp)
```

```
# efskeymgr -k user/user5 -S group/group1
$ id
uid=207(user5) gid=1(staff) groups=202(group1)
$ efskeymgr -v
 Keystore content:
      Keystore owner ............ : uid 207
      Keystore mode ............. : admin: managed by EFS administrator
      Password changed last on .. : 05/03/07 at 17:39:43
    Private key:
          Algorithm : RSA_1024
        Fingerprint : 49e42532:e8c471bf:2be9ba03:3f9d12b4:4ba01cea
           Validity : This key is valid.
$ efskeymgr -V
List of keys loaded in the current process:
 Key #0:
    Kind ..................... User key
    Id   (uid / gid) ........ 207
    Type .................... Private key
    Algorithm ............... RSA_1024
    Validity ................ Key is valid
    Fingerprint .............
49e42532:e8c471bf:2be9ba03:3f9d12b4:4ba01cea
 Key #1:
    Kind ..................... Group key
    Id   (uid / gid) ........ 202
    Type .................... Private key
    Algorithm ............... RSA_1024
    Validity ................ Key is valid
    Fingerprint .............
5100b522:fbee1dbe:2aeddbd3:c2e51fea:86ab4b5c
```

Important: When a user is removed from a group, either using **chuser** or **chgroup**, the group access key is not automatically deleted from the user keystore.

2.6.6 Adding/remove group access keys in root guard mode

When running in root guard mode, root cannot add himself as a member of the group. When a group member is added/removed to/from a group by the group administrator, a corresponding cookie is sent to user keystore.

The user is prompted to accept/decline the group access key first time he or she opens his or her keystore, as shown in Example 2-42 on page 141.

Example 2-42   Accepting/declining group access keys in root guard mode

```
$ efskeymgr -v

The following operation(s) is(are) pending on your EFS keystore:

        You are granted access to group/group2 keystore.
Do you want to process this action now (y), later (n: default), or
never (d)? [ y / n / d ]n
        You are removed access to group/group2 keystore.
Do you want to process this action now (y), later (n: default), or
never (d)? [ y / n / d ]n
 Keystore content:
     Keystore owner ............ : uid 219
     Keystore mode ............. : guard: not managed by EFS
administrator
     Password changed last on .. : 05/13/07 at 18:30:03
    Private key:
         Algorithm : RSA_1024
       Fingerprint : 1d0b0ee6:00ec743a:db95715c:ce04ff89:0b1ecfea
          Validity : This key is valid.
Cookie:
  Type ...... : Remove (removes access to)
  Keystore .. : group/group2
Cookie:
  Type ...... : Access (grants access to)
  Keystore .. : group/group2
```

2.6.7 Managing a group keystore private key

When running in root admin mode, root can manage the group keystore as shown in the scenario in Example 2-43 on page 142 as follows:

► The `lsgroup` commands show that user3 belongs to group1.

► The `efskeymgr -v -k group/group1` command displays the fingerprint of the group key.

► user3 has the group key loaded in the current shell.

► Root generates a new group key. The old key is still kept in the group keystore and is marked as deprecated.

► The new key has automatically been sent to the user keystore. The old group key is marked as deprecated in the user keystore. The new group key will be the active key in all new processes.

► Root deletes the deprecated key from the group keystore.

► The deprecated key has been deleted from the user keystore, but not unloaded form active processes, the current shell in our case. The newly-generated shell does no contain the old key any more

Example 2-43   Changing group keystore private key

```
# lsgroup -a users group1
group1 users=user1,user3
# efskeymgr -v -k group/group1
 Keystore content:
     Keystore owner ............ : gid 202
     Keystore mode ............. : admin: managed by EFS administrator
     Password changed last on .. : 05/09/07 at 19:43:52
   Private key:
         Algorithm : RSA_1024
       Fingerprint : 5967aba6:df046b56:8f4cc1b5:69f48c7a:ed88008f
          Validity : This key is valid.
$ id
uid=205(user3) gid=1(staff)
$ efskeymgr -v
 Keystore content:
     Keystore owner ............ : uid 205
     Keystore mode ............. : admin: managed by EFS administrator
     Password changed last on .. : 05/10/07 at 09:41:20
   Private key:
         Algorithm : RSA_1024
       Fingerprint : 44b0c7e1:53384985:ca1be27e:864b9431:3a57c0d1
          Validity : This key is valid.
   Access key to keystore group/group1
$ efskeymgr -V
List of keys loaded in the current process:
 Key #0:
   Kind .................... User key
   Id   (uid / gid) ........ 205
   Type ................... Private key
   Algorithm ............... RSA_1024
   Validity ................ Key is valid
   Fingerprint .............
44b0c7e1:53384985:ca1be27e:864b9431:3a57c0d1
 Key #1:
   Kind .................... Group key
   Id   (uid / gid) ........ 202
   Type ................... Private key
   Algorithm ............... RSA_1024
   Validity ................ Key is valid
```

```
      Fingerprint ..............
5967aba6:df046b56:8f4cc1b5:69f48c7a:ed88008f
# efskeymgr  -k group/group1 -R RSA_1024
# efskeymgr -v -k group/group1
 Keystore content:
      Keystore owner ............ : gid 202
      Keystore mode ............. : admin: managed by EFS administrator
      Password changed last on .. : 05/09/07 at 19:43:52
    Private key:
          Algorithm : RSA_1024
        Fingerprint : 9f5f057d:4f3f5a78:50058cd3:bf1a2e78:42ae3b20
           Validity : This key is valid.
    Private key:
          Algorithm : RSA_1024
        Fingerprint : 5967aba6:df046b56:8f4cc1b5:69f48c7a:ed88008f
           Validity : This key was deprecated on 05/10/07 at 20:15:53.
$ id
uid=205(user3) gid=1(staff)
$ efskeymgr -o ksh
$ efskeymgr -V
List of keys loaded in the current process:
 Key #0:
    Kind .................... User key
    Id   (uid / gid) ........ 205
    Type .................... Private key
    Algorithm ............... RSA_1024
    Validity ................ Key is valid
    Fingerprint ..............
44b0c7e1:53384985:ca1be27e:864b9431:3a57c0d1
 Key #1:
    Kind .................... Group key
    Id   (uid / gid) ........ 202
    Type .................... Private key
    Algorithm ............... RSA_1024
    Validity ................ Key is deprecated
    Fingerprint ..............
5967aba6:df046b56:8f4cc1b5:69f48c7a:ed88008f
 Key #2:
    Kind .................... Group key
    Id   (uid / gid) ........ 202
    Type .................... Private key
    Algorithm ............... RSA_1024
    Validity ................ Key is valid
    Fingerprint ..............
9f5f057d:4f3f5a78:50058cd3:bf1a2e78:42ae3b20
```

```
# efskeymgr -k group/group1 -D
5967aba6:df046b56:8f4cc1b5:69f48c7a:ed88008f
# efskeymgr -v -k group/group1
 Keystore content:
     Keystore owner ............ : gid 202
     Keystore mode ............. : admin: managed by EFS administrator
     Password changed last on .. : 05/09/07 at 19:43:52
   Private key:
         Algorithm : RSA_1024
       Fingerprint : 9f5f057d:4f3f5a78:50058cd3:bf1a2e78:42ae3b20
         Validity : This key is valid.
$ id
uid=205(user3) gid=1(staff)
$ efskeymgr -o ksh
$ efskeymgr -V
List of keys loaded in the current process:
 Key #0:
   Kind ................... User key
   Id   (uid / gid) ........ 205
   Type .................. Private key
   Algorithm .............. RSA_1024
   Validity ............... Key is valid
   Fingerprint ...........
44b0c7e1:53384985:ca1be27e:864b9431:3a57c0d1
 Key #1:
   Kind ................... Group key
   Id   (uid / gid) ........ 202
   Type .................. Private key
   Algorithm .............. RSA_1024
   Validity ............... Key is valid
   Fingerprint ...........
9f5f057d:4f3f5a78:50058cd3:bf1a2e78:42ae3b20
```

When running in root guard mode, only a group administrator can generate a
new group access key, deprecate a key, or delete a deprecated key.

2.6.8 Sending/removing the group keystore access key to/from another group keystore

When running in root admin mode, you can use the **efskeymgr** command to send the access key for a group store to a user group keystore using the following steps as shown in Example 2-44:

► The **efskeymgr** command is used twice to display the content of the group keystore for group1 and group2.

► user3 is a member of group2, as shown by the **efskeymgr -v** command.

► Root sends the access key of group1 to the group2 keystore using the **efskeymgr -s** command.

► The **efskeymgr -v -k group/group2** command shows that the group2 keystore contains the access key of the group1 keystore.

► The content of user3 keystore has not changed because the user has not been directly granted any additional access

► When the content of the user3 keystore is reopened using the **efskeymgr -o ksh** command, the newly-generated shell gets the group access key for group1 as well.

► Root removes the access key of group1 from the group2 keystore using the **efskeymgr -S** command.

► The **efskeymgr -v -k group/group2** command shows that the group2 keystore no longer contains the access key of the group1 keystore.

► When the content of user3 keystore is reopened using the **efskeymgr -o ksh** command, the newly-generated shell no longer has the group1 access key.

Example 2-44   Sending a group keystore access key to another group

```
# efskeymgr -v -k group/group1
 Keystore content:
     Keystore owner ............ : gid 204
     Keystore mode ............. : admin: managed by EFS administrator
     Password changed last on .. : 05/13/07 at 13:17:25
   Private key:
         Algorithm : RSA_1024
       Fingerprint : 53cd3824:7c9d508e:b825a253:f3209fcf:76f1766f
          Validity : This key is valid.
# efskeymgr -v -k group/group2
 Keystore content:
     Keystore owner ............ : gid 205
     Keystore mode ............. : admin: managed by EFS administrator
     Password changed last on .. : 05/13/07 at 13:51:07
```

```
        Private key:
            Algorithm : RSA_1024
          Fingerprint : 61e6538d:343c8313:df807f20:ab02451e:734309ca
             Validity : This key is valid.
$ id
uid=212(user3) gid=1(staff)
$ efskeymgr -v
 Keystore content:
     Keystore owner ............ : uid 212
     Keystore mode ............. : admin: managed by EFS administrator
     Password changed last on .. : 05/13/07 at 13:48:07
     Private key:
            Algorithm : RSA_1024
          Fingerprint : 1c9e287c:05a6b7b3:020ce4c5:0f578619:5c7ea3b6
             Validity : This key is valid.
     Access key to keystore group/group2
# efskeymgr -k group/group1 -s group/group2
# efskeymgr -v -k group/group2
 Keystore content:
     Keystore owner ............ : gid 205
     Keystore mode ............. : admin: managed by EFS administrator
     Password changed last on .. : 05/13/07 at 13:51:07
     Private key:
            Algorithm : RSA_1024
          Fingerprint : 61e6538d:343c8313:df807f20:ab02451e:734309ca
             Validity : This key is valid.
     Access key to keystore group/group1
$ id
uid=212(user3) gid=1(staff)
$ efskeymgr -v
 Keystore content:
     Keystore owner ............ : uid 212
     Keystore mode ............. : admin: managed by EFS administrator
     Password changed last on .. : 05/13/07 at 13:48:07
     Private key:
            Algorithm : RSA_1024
          Fingerprint : 1c9e287c:05a6b7b3:020ce4c5:0f578619:5c7ea3b6
             Validity : This key is valid.
     Access key to keystore group/group2
$ efskeymgr -V
List of keys loaded in the current process:
 Key #0:
   Kind .................... User key
   Id   (uid / gid) ........ 212
   Type .................... Private key
```

```
   Algorithm ................ RSA_1024
   Validity ................ Key is valid
   Fingerprint ..............
61c9e287c:05a6b7b3:020ce4c5:0f578619:5c7ea3b6
 Key #1:
   Kind .................... Group key
   Id   (uid / gid) ........ 204
   Type .................... Private key
   Algorithm ................ RSA_1024
   Validity ................ Key is valid
   Fingerprint ..............
53cd3824:7c9d508e:b825a253:f3209fcf:76f1766f
 Key #2:
   Kind .................... Group key
   Id   (uid / gid) ........ 205
   Type .................... Private key
   Algorithm ................ RSA_1024
   Validity ................ Key is valid
   Fingerprint ..............
61e6538d:343c8313:df807f20:ab02451e:734309ca
# efskeymgr -k group/group2 -S group/group1
# efskeymgr -v -k group/group2
 Keystore content:
     Keystore owner ............ : gid 205
     Keystore mode ............ : admin: managed by EFS administrator
     Password changed last on .. : 05/13/07 at 13:51:07
   Private key:
         Algorithm : RSA_1024
       Fingerprint : 61e6538d:343c8313:df807f20:ab02451e:734309ca
          Validity : This key is valid.
$ id
uid=212(user3) gid=1(staff)
$ efskeymgr -o ksh
$ efskeymgr -V
List of keys loaded in the current process:
 Key #0:
   Kind .................... User key
   Id   (uid / gid) ........ 212
   Type .................... Private key
   Algorithm ................ RSA_1024
   Validity ................ Key is valid
   Fingerprint ..............
1c9e287c:05a6b7b3:020ce4c5:0f578619:5c7ea3b6
 Key #1:
   Kind .................... Group key
```

```
   Id   (uid / gid) ......... 205
   Type .................... Private key
   Algorithm ............... RSA_1024
   Validity ................ Key is valid
   Fingerprint ..............
61e6538d:343c8313:df807f20:ab02451e:734309ca
```

2.7 Back up and restore

The `backup` command has a new flag that allows backups of encrypted files or
encrypted file systems. Along with the data from files or a file system, all
cryptographic metadata is also backed up. At restore time, this information is
restored as well. You must also be able to access this information, so you also
must back up any user keystore passwords.

> **Important:** When running in root admin mode, the root user can reset the
> password of a user keystore. When running in root guard mode, root cannot
> get access to a user keystore, so you must save the user keystore password
> as well.

To summarize, any viable backup should include:

► File or file system backup
► Keystores of all users that are the owners of the files being backed up
► Group keystores
► Passwords for all user keystores

2.7.1 Backing up encrypted files

In this section, we show:

► How you can back up both encrypted and cleartext file in the same archive.
► How access to an encrypted file is required in order to perform backup.
► How the internal format of the archive is dependent on the archive location.
► That the archive itself is in encrypted format.
► How the archive format can efficiently protect the information contained in the
  archive.

In Example 2-45, we show how you can back up your files as follows:

- In the encrypted file systems /efs, there are two files named file_encrypted and file_in_clear, one in encrypted format, the other in cleartext format.

- The information contained in both files that are used in this scenarios is displayed.

- The keys from the current shell are unloaded using the `efskeymgr -c ksh` command.

- user1 tries to archive both files in an archive named tmp_archive located on the non-encrypted file system /tmp using the `ls file*|backup -ivZf /tmp/tmp_archive` command. The command fails to open the encrypted file. However, the archive is created. Searching in the contents of the archive using the `cat /tmp/tmp_archive` command reveals the name of the file file_in_clear and the content of the file as well.

- user1 tries to archive both files in a archive named efs_archive located on the encrypted file system /efs using the `ls file*|backup -ivZf /efs/tmp_archive` command. The command fails and no archive is created.

- user1 must load the content of the keystore in the current shell.

- user1 tries to archive both files in an archive named efs_archive located on the encrypted file system /efs using the `ls file*|backup -ivZf /efs/tmp_archive` command. The command completes successfully and the archive is created.The archive itself is an encrypted file.

- user1 tries to archive both files in a archive named tmp_archive located on the non-encrypted file system /tmp using the `ls file*|backup -ivZf /tmp/tmp_archive` command. The command completes successfully and the archive is created. However, the archive itself is an not an encrypted file.

- user2 logs in. It has access to the /efs file system, but cannot access the file efs_archive.

- user2 has access to the /tmp file system and to the file tmp_archive. Searching in the content of the files reveals the name of the file file_in_clear and the content of the file as well. The content of the encrypted file has not been disclosed.

Example 2-45   File backup

```
user1@guadalupe/]id
uid=205(user1) gid=1(staff) groups=202(group1)
user1@guadalupe/]ls -alU
total 48
drwxrwxrwxe    3 root     system         4096 May 07 20:52 .
drwxr-xr-x-   20 root     system         4096 May 04 15:46 ..
-rw-r--r--e    1 user1    staff            15 May 07 19:35 file_encrypted
-rw-r--r---    1 user1    staff            11 May 07 19:36 file_in_clear
drwxr-xr-x-    2 root     system          256 May 03 15:46 lost+found
```

```
user1@guadalupe/]cat file_encrypted
encrypted text
user1@guadalupe/]cat file_in_clear
cleartext
user1@guadalupe/]efskeymgr -c ksh
user1@guadalupe/]ls file*|backup -ivZf /tmp/tmp_archive
Mount volume 1 on /tmp/tmp_archive.
        Press Enter to continue.
Backing up to /tmp/tmp_archive.
Cluster 51200 bytes (100 blocks).
Volume 1 on /tmp/tmp_archive
backup: 0511-449 An error occurred accessing file_encrypted: Cannot find the
requested security attribute.
a           11 file_in_clear
The total size is 11 bytes.
Backup finished on Mon May  7 21:04:49 CDT 2007; there are 100 blocks on 1 volumes.
user1@guadalupe/]cat /tmp/tmp_archive
        kêäÀÚ?FÃÚ?Fÿÿby nameby namerootd


kêtõ
    ¤Í
      RÚ?FÆ?FÆ?F


file_in_clearlc¤clear text
kê


user1@guadalupe/]ls file*|backup -ivZf /efs/efs_archive
Mount volume 1 on /efs/efs_archive.
        Press Enter to continue.
backup: 0511-089 Cannot open /efs/efs_archive: Cannot find the requested security
attribute.
Mount volume 1 on /efs/efs_archive.
user1@guadalupe/]efskeymgr -o ksh
user1's EFS password:
user1@guadalupe/]ls file*|backup -ivZf /efs/efs_archive
Mount volume 1 on /efs/efs_archive.
        Press Enter to continue.
Backing up to /efs/efs_archive.
Cluster 51200 bytes (100 blocks).
Volume 1 on /efs/efs_archive
a         4096 file_encrypted
a           11 file_in_clear
The total size is 4107 bytes.
```

```
Backup finished on Mon May  7 21:18:45 CDT 2007; there are 100 blocks on 1 volumes.
user1@guadalupe/]efsmgr -l efs_archive
EFS File information:
 Algorithm: AES_128_CBC
List of keys that can open the file:
 Key #1:
  Algorithm      : RSA_1024
  Who            : uid 205
  Key fingerprint : 92441557:db77320c:daa7e5d4:cf2877ce:30c60454
user1@guadalupe/]ls file*|backup -ivZf /tmp/tmp_archive
Mount volume 1 on /tmp/tmp_archive.
        Press Enter to continue.
Backing up to /tmp/tmp_archive.
Cluster 51200 bytes (100 blocks).
Volume 1 on /tmp/tmp_archive
a          4096 file_encrypted
a            11 file_in_clear
The total size is 4107 bytes.
Backup finished on Mon May  7 21:23:04 CDT 2007; there are 100 blocks on 1 volumes.
user1@guadalupe/]echo $?
0
user2@guadalupe/]id
uid=204(user2) gid=1(staff) groups=202(group1)
user2@guadalupe/]cat efs_archive
cat: 0652-050 Cannot open efs_archive.
user2@guadalupe/]cat tmp_archive
        kêýß?Fß?Fÿÿby nameby namerootd
                                      kêªÑ¤HÍàÞ?FØÅ?FØÅ?F


file_encrypted×èÁµE "¥î¾¸¡ÐJób)½ÏÔã
 uØÄ¤                              {¶!zùb-PÂy'[Gf«ÈUgXOu%;´kg


    Ë)Ö/#
ïÜîxÇÜçPrôbë
:íqöÅS ª´-"?`H[7ásVÅ7ü


æ¸âþ»¦ºn#NrÛôðw½ÏæQYkK.ÆÆwSE«-¦F|º¯}KbÃo]ýßòcI,£ÕpI¼ÛØÉ©LrïäC<rù×ªkà;ûXªñïM8æ@
×êºA ÜkþA2Æ»    `Ö2]ç
jnÝKiåêÿ3æùÎ¯+¢67\ä µgzÕtÁäNÆqZ|n\¯ X§"EÚ4RpÜ¨éV
sdmå¸î1±1ÃRÌ¯ tûHõ_ñ&M÷ãâI
í Ì°+æ)ÊÄ}uÁÅgÊÛ÷ùOO]ÎcÃ 4^OãAaÐ*¨;\]$÷ÝòÎ)&o5¶R+rhaì%ïGÑhºÉÇ¾e@ªÝ4ÔÁceö`WÙ'¿jxNhq?$î
ÞË¸`
        ÍáDÌÓé6
                VÔdôwé
                        Í ßXªQ8RÆ=õ9çÄ`J& /
```

ý!#,uÑ)|

±¿=ÔbÃ¨Û¨{ÆþgëS:øç_35 qrS_|ór¥ÚîÍ½¨°Ñpë#âêª<ûH æ¿-?P§qæ½'âÝZ[W

1V÷ÈäÏ¯kÖM>ß<#,Ç¶

Jãëµ °Ì 5?9Pöu¯´dîü9åq*µßGK ãÜ¯r#ëtÊd ùVîË÷¸Þ{«¢¤ALù

¿f

áZð ó¾Å·±ÅqÊK±°M3þ?$Ï ÑzõxqÜÐ5=ei\îg¦»ý.UîîÂ>öä):ÝÂ,6"Hò6©¶í

Ký/ÆJf¿[cÔf5ln 9æÒwºx

Ô-Ì ~-Z`)Sq÷ÜÃþçþôÇñ.K_|éû Îxc£X¨òÜñëðã¯_©ÖHS¯íGæ

qÌNéP HÕÔ9}-µýÚr!¾~©BU?[

46!7Õ]Fôò±T

fq,ô,·ÇêÙÕ Gzõ/2‰Þ{¾å-õ'e@hèUÛîÜä (6P¨íúÏ@

»J)R±(1¾QÕa¶e

´jïÃ)ðIÅ   õ¸¯Íço(ßÎÏzÏ?úÔSGZô' D2

]Æ4YÛË´·êp÷o4ñüg¨@¶þôêdêôâ⁴8ÈÛÆK5l©5}/ðá|E$3Õ ÷ýÏp

©íô¥êéáøäD¬Þ)sqÍ©£ì¯º5?TV§çþÞ4µ   ä*K.Ãí

joP½àkYbīz<]°fñ<¨=\M#Ô.ú*rä¤Ñ

`ÔäHxs×uZ`,¸OlY¯=¢Õ¥ÿæäÕß   ¾ÅGýÚlpāÚ-±QW_P_~ÛjÃ£eºTi/EMI

ËûÔQ·Fk2=AaQèÉz{A»É¾ÍRV«þç$ÞæíØxÖÛ¸\b

»ÿô¯[¿ôNà-õú¸#Ð6(Ïuóã~fíF*Ã¤xÙï0Ä*P+Û5*JóHïurv

¢¬1MzâØ1ÃÆ4f"á,ôöGÈP÷«Müþï?kªktTµ+"Yq¾º6o ©<^rÔ¯¯Au

xjô¨: VpÿÂrèG[m9|Eú-jËFâÛáPkðëå±Ã[b2ç±,¦Ü‰Û-ñÆv¼u

Q´Äāl± SLÄÕ-kMâôî9z-?1ÖVÉ¯T»Gv'{8C^+/× "¶Ì¼_Ôx-ÜWÐV

økBTÙÿñPs±_¦;KÑåõ{uÄÂò¬

+HãÊß½ZJUc#@ùw5RÛ!Õ@+¼Nhīh}EI åÇyMb$N¥e´"

 N@ë?dnÅdè!n7jüp&È0îÀPCò~âT¿þÙoraE¤jÊõçþ@F¤ÅVø]úî£#÷Ã*ÈÚ¸u-V?Ãé=

êÂSÊ3-#Jÿ<t^wÞmA euaÇÚâúçk@ Ü   Â´Þä© É÷Á÷`Äpn4ZQèïM

ã^£/]ûªUÅQg-=;5½Î

¤

ÐDÜ¦ºþ.KÈe¾9 ÞY~ï Z:^éT &ÊH£WxÅd«5D   V‰"ñÈ4ko oëk

«ēúK,µDÒnV#   !û/<â`ÌC¢\gi'·åO

SÜgª;dÃçtX$6soÊ"·í-¬ëiÖ km Wzè8ãâiSÀÍh´6i}B

7fë1óÐÔ

á

fÆ"Ã-úïX¥RøjD¾QZÜZ1 "

Äºt °vÏDī{i!}vÖiCÅTvhVtg]ÄÖµMcpA뢠fíbÎ©¢A Øo°ÄÁFHôøi'VXKG@¯ÜyK?A«=|ÜÄÙJ}ÙAù#þ

ö ñ÷ KwÔÿê¶\VGwIīnÁßE

-Æ¶jû- !þ)lñóîå"Vò!Éº×=îWBª×NÊ±9[¼î9ÎÊ¯(/W

Jlô÷ÈÊÑ½ðÂºGM-s!ac`¥ñ~§nÂyÆÉ+Þþ|hÔ+

-c^ÿÍþÿé{"ü

1lÿWÔ±Ë

´ÛãÅqv>1Ykeó

Ê~±'[/tc7ñý5Koµ ÏÜÿ¢ò

ao
          Ngp©$
ÛOÜeW!y
"C
èS&60¶Ú?%°Z¶Z_ÙãÆ>¥ø=½Z%àrV$*!P!§Á×j bu©
                                        ¿ÃsØYyÀ'Ñ1nQÁoçDïYµÊúÏY3Iún2Å -ØR
                                                            «[§ÅË'{Fõ
x)ç¨0]ù|zï.täçc¬ÇÆ$½ÔÕåzY¿=ö¸ Éù×\ÜÌEÚñÆ]°¥gù/Å/ÙMþÏ
ÚÝ 1z±§¶)ñ6löi9ôÏí5!µÖTÁL3«¨2   ì+ZQx> E3'D7IÄQyÆx
                                                        d¢Xvj¾#
í¢^ÙL´>þ4Ìû^áxüZ¼îôËÛCÝ¾ uãôÖj,ÿ(ÑÔÃ~#_t¶J3\U
OCÆyS9R£ ý};ì«·X7ÛÞÆ?õLïºR6 j¯×ëùÂiºmûgäZ¶§ Ç÷Öqü.Aó¿ÆUÀ`×*{  1ßôÂ¢$++ùo,¸#LQêç)`
@~ÿÿ&Ù |ÅYH
               /ñé@8íÜ[UEú£SL
^?ìÈ+¤ÿ`Û mA
          }Äôð!Ö=K¢h YúÑ~"M?VE¾ÂÔQåxzo ËqGÃ=ÌÙ á×
k lgÇMýâ"D ¦,¤
Ñ
 äÛ9)ëæ:zSqç g>v~+¾H"ÏZÆ±{Õ,¢&e O!RQÔ#âiùfÑÅùìÏÔÊ?8ÐÃC=ØYÇÉd?
Ö¶ç!PÝËSÜ))Ú¤÷ïëhôÙ¦jùÆ*Eý± èçMÈãGV©   *ïõCTºÎ´¿   iYE
4ê¯p0
+ÝÖ`Õra
       GåPïmÃRZ*\«C»´_2}'Ö.ØûPiau¥}¶o2ßá[\Ã](»¶6RR
                              vD©nqBe3ÖJ(Y"äa1ã02µSK¼÷#¸Ö]
                                                    [ý¾0
Ï!7
°zy½BÂ÷#úÓN~¡ªlÉ%Kç»Kõ
{
                              ´º@-úÔcDæl¦Ü¯ðå
kêD¶EFSFILESIZEDone15Ð4ô~þN~|8ÛÓÙÙ¿
kêÔ¤ß?FØÅ?FØÅ?F
øSYSTEMø_NREêñsÍDWÛw2
Ú§åÒÏ(wÎ0ÆT;a;óîFF+çWnùIz@0Á6êèHÊÕÂÔ´*ÂF
                              *?·s¢ß49ìÙQé hÆ&¼@[¶*Yø
Ãri7ÂÏ/±,6t_×§[}t+ ´È/Îà
               ïôÿ

               kê$)
                    ¤Í
                    àÞ?FÆ?FÆ?F

file_in_clearlc¤clear text
,ikê

```
PuTTYPuTTYPuTTYPuTTYPuTTYPuTTYPuTTYPuTTYPuTTYPuTTYPuTTYPuTTYPuTTYPuTTYPuTTYPuTTY
user2@guadalupe/]PuTTYPuTTYPuTTYPuTTYPuTTYPuTTYPuTTYPuTTYPuTTYPuTTY    ksh:
PuTTYPuTTYPuTTYPuTTYPuTTYPuTTYPuTTYPuTTYPuTTYPuTTYPuTTYPuTTYPuTTYPuTTYPuTTYPuTTY
:  not found.
```

2.7.2 Restoring encrypted files

Here we discuss restoring encrypted files.

Scenario 1

We try to restore the archives created in Example 2-46 in a encrypted file systems named /jfs2 as follows:

▶ user1 successfully copies efs_archive and tmp_archive in /efs2. Both files are encrypted, although originally the file tmp_acrhive was in cleartext.

▶ Restoring the efs_archive is successful. Both original files can be accessed and displayed.

▶ Restoring the tmp_archive is successful. Both original files can be accessed and displayed

Example 2-46   Restoring files in an encrypted file system

```
user1@guadalupe/]ls -alU
total 240
drwxrwxrwxe    3 root     system          256 May 07 22:15 .
drwxr-xr-x-   22 root     system         4096 May 07 22:07 ..
-rw-r--r--e    1 user1    staff         51200 May 07 22:15 efs_archive
drwxr-xr-x-    2 root     system          256 May 07 22:07 lost+found
-rw-r--r--e    1 user1    staff         51200 May 07 22:15 tmp_archive
user1@guadalupe/]restore -xvf efs_archive
Please mount volume 1 on efs_archive.
        Press the Enter key to continue.

New volume on efs_archive:
Cluster size is 51200 bytes (100 blocks).
The volume number is 1.
The backup date is: Mon May  7 21:18:44 CDT 2007
Files are backed up by name.
The user is root.
x        4096 file_encrypted
x          11 file_in_clear
The total size is 4107 bytes.
The number of restored files is 2.
user1@guadalupe/]ls -alU
```

```
total 272
drwxrwxrwxe    3 root      system           256 May 07 22:17 .
drwxr-xr-x-   22 root      system          4096 May 07 22:07 ..
-rw-r--r--e    1 user1     staff          51200 May 07 22:15 efs_archive
-rw-r--r--e    1 user1     staff             15 May 07 19:35
file_encrypted
-rw-r--r--e    1 user1     staff             11 May 07 19:36
file_in_clear
drwxr-xr-x-    2 root      system           256 May 07 22:07 lost+found
-rw-r--r--e    1 user1     staff          51200 May 07 22:15 tmp_archive
user1@guadalupe/]cat file_in_clear
clear text
user1@guadalupe/]cat file_encrypted
encrypted text
user1@guadalupe/]rm file*
user1@guadalupe/]restore -xvf tmp_archive
Please mount volume 1 on tmp_archive.
        Press the Enter key to continue.


New volume on tmp_archive:
Cluster size is 51200 bytes (100 blocks).
The volume number is 1.
The backup date is: Mon May  7 21:23:03 CDT 2007
Files are backed up by name.
The user is root.
x          4096 file_encrypted
x            11 file_in_clear
The total size is 4107 bytes.
The number of restored files is 2.
user1@guadalupe/]ls -alU
total 272
drwxrwxrwxe    3 root      system           256 May 07 22:20 .
drwxr-xr-x-   22 root      system          4096 May 07 22:07 ..
-rw-r--r--e    1 user1     staff          51200 May 07 22:15 efs_archive
-rw-r--r--e    1 user1     staff             15 May 07 19:35
file_encrypted
-rw-r--r--e    1 user1     staff             11 May 07 19:36
file_in_clear
drwxr-xr-x-    2 root      system           256 May 07 22:07 lost+found
user1@guadalupe/]cat file_in_clear
clear text
user1@guadalupe/]cat file_encrypted
encrypted text
```

Scenario 2

In this scenario, we show how you can restore files backed up by a different user than the file owner, as shown in Example 2-47:

► The file is owned by user2. He or she is the only user that can access the file. Not even root can access the file and archive it.

► Access is granted to root to the file.

► The **backup** command succeeds. The archive file is owned by root.

► The **restore** command is successful. File attributes and access keys are restored successfully.

► Access is granted for user5.

► The **backup** command succeeds. The archive file is owned by user5.

► The **restore** command is successful. The file that has been created has the same permissions and the same access keys, but is owned now by user5.

Example 2-47   Restoring files owned by a different user

```
user2@guadalupe/]ls -alU
total 40
drwxrwxrwxe    3 root      system         4096 May 08 12:22 .
drwxr-xr-x-   22 root      system         4096 May 07 22:07 ..
-rw-r--r--e    1 user2     staff            13 May 08 12:17 file
drwxr-xr-x-    2 root      system          256 May 03 15:46 lost+found
user2@guadalupe/]efsmgr -l file
EFS File information:
 Algorithm: AES_128_CBC
List of keys that can open the file:
 Key #1:
  Algorithm      : RSA_1024
  Who            : uid 204
  Key fingerprint : 047dcbd4:f7528784:44488eab:7420d4f2:af3e8bee
root@guadalupe/]ls file|backup -ivZf archive
Mount volume 1 on archive.
       Press Enter to continue.
Backing up to archive.
Cluster 51200 bytes (100 blocks).
Volume 1 on archive
backup: 0511-449 An error occurred accessing file: Security
authentication is denied.
The total size is 0 bytes.
Backup finished on Tue May  8 12:34:41 CDT 2007; there are 100 blocks
on 1 volumes.
user2@guadalupe/]efsmgr -a file -u root
```

```
root@guadalupe/]ls file|backup -ivZf archive
Mount volume 1 on archive.
        Press Enter to continue.
Backing up to archive.
Cluster 51200 bytes (100 blocks).
Volume 1 on archive
a        4096 file
The total size is 4096 bytes.
Backup finished on Tue May  8 12:32:55 CDT 2007; there are 100 blocks
on 1 volumes.
root@guadalupe/]restore -xvf archive
Please mount volume 1 on archive.
        Press the Enter key to continue.


New volume on archive:
Cluster size is 51200 bytes (100 blocks).
The volume number is 1.
The backup date is: Tue May  8 13:37:16 CDT 2007
Files are backed up by name.
The user is root.
x        4096 file
The total size is 4096 bytes.
The number of restored files is 1.
root@guadalupe/]ls -alU file
-rw-r--r--e   1 user2    staff              13 May 08 13:34 file
root@guadalupe/]efsmgr -l file
EFS File information:
 Algorithm: AES_128_CBC
List of keys that can open the file:
 Key #1:
  Algorithm      : RSA_2048
  Who            : uid 0
  Key fingerprint : 3ea8581e:16982aa8:dfec5c5a:f02251f1:cfca0445
 Key #2:
  Algorithm      : RSA_1024
  Who            : uid 204
  Key fingerprint : 047dcbd4:f7528784:44488eab:7420d4f2:af3e8bee
user2@guadalupe/]efsmgr -a file -u user5
user2@guadalupe/]efsmgr -l file
EFS File information:
 Algorithm: AES_128_CBC
List of keys that can open the file:
 Key #1:
  Algorithm      : RSA_1024
  Who            : uid 204
```

```
   Key fingerprint : 047dcbd4:f7528784:44488eab:7420d4f2:af3e8bee
 Key #2:
  Algorithm      : RSA_1024
  Who            : uid 207
  Key fingerprint : 49e42532:e8c471bf:2be9ba03:3f9d12b4:4ba01cea
user5@guadalupe/]ls file|backup -ivZf archive_user
Mount volume 1 on archive_user.
        Press Enter to continue.
Backing up to archive_user.
Cluster 51200 bytes (100 blocks).
Volume 1 on archive_user
a        4096 file
The total size is 4096 bytes.
Backup finished on Tue May  8 13:50:16 CDT 2007; there are 100 blocks
on 1 volumes.
user5@guadalupe/]ls -al
total 152
drwxrwxrwx   3 root     system         4096 May 08 13:50 .
drwxr-xr-x  22 root     system         4096 May 07 22:07 ..
-rw-r--r--   1 user5    staff         51200 May 08 13:50 archive_user
-rw-r--r--   1 user2    staff            13 May 08 13:34 file
drwxr-xr-x   2 root     system          256 May 03 15:46 lost+found
user5@guadalupe/]restore -xvf archive_user
Please mount volume 1 on archive_user.
        Press the Enter key to continue.


New volume on archive_user:
Cluster size is 51200 bytes (100 blocks).
The volume number is 1.
The backup date is: Tue May  8 13:50:16 CDT 2007
Files are backed up by name.
The user is root.
x        4096 file
The total size is 4096 bytes.
The number of restored files is 1.
user5@guadalupe/]ls -al
total 168
drwxrwxrwx   3 root     system         4096 May 08 13:51 .
drwxr-xr-x  22 root     system         4096 May 07 22:07 ..
-rw-r--r--   1 user5    staff         51200 May 08 13:50 archive_user
-rw-r--r--   1 user5    staff            13 May 08 13:34 file
drwxr-xr-x   2 root     system          256 May 03 15:46 lost+found
user5@guadalupe/]efsmgr -l file
EFS File information:
 Algorithm: AES_128_CBC
```

```
List of keys that can open the file:
 Key #1:
  Algorithm      : RSA_1024
  Who            : uid 204
  Key fingerprint : 047dcbd4:f7528784:44488eab:7420d4f2:af3e8bee
 Key #2:
  Algorithm      : RSA_1024
  Who            : uid 207
  Key fingerprint : 49e42532:e8c471bf:2be9ba03:3f9d12b4:4ba01cea
```

2.7.3 User private keys impact on file backup/restore

In this scenario, we show that information contained in user keystores is vital for successful restoration of an archive. User and group keystores must also be backed up in order to be able to access the files that are restored. You must also ensure access to user and group keystores.

A practical backup/restore scenario is shown in Example 2-48 on page 160 as follows:

► user6 creates file1 using the `echo` command.

► The `efsmgr -l` command shows that file1 can be accessed using the private key whose fingerprint is c566b5cf:04921ffa:1480a2bc:b34d48f3:c9cccb9c. For simplicity, we will refer it as key1.

► The `efskeymgr -v` command shows that key1 is stored in the user keystore.

► user6 generates a new private key named key2 using the `efskeymgr -R` command.

► user6 loads the content of the keystore in the current shell so key2 will be used instead of key1.

► user6 creates file2 using the `echo` command.

► Using the `efsmgr -l` command, the fingerprint of key2 is ffcbef78:cf865d70:bfab5101:ab572d70:f3326d79.

► user6 generates a new private key named key3 using the `efskeymgr -R` command.

► user6 loads the content of the keystore in the current shell so key3 will be used instead of key2.

► user6 creates file3 using the `echo` command.

► Using the `efsmgr -l` command, the fingerprint of key3 is 340e3449:a1fb5712:6163de83:02c08db1:400d830e.

- ► The **efskeymgr -v** command shows that key3 is active and key1 and key2 have been deprecated. However, they are still present in the user keystore.

- ► user6 backs up file1, file2, and file3. He or she can add all files to the archive because he or she still has the deprecated keys. The archive itself is an encrypted file that can be accessed with key3.

- ► The original file1, file2, and file3 are removed.

- ► user6 restores all files using the **restore** command. All restored files preserve the access key at the time they were created.

- ► user6 still has access to all files as shown by the **cat** command.

- ► key2 is deleted and removed from the user6 keystore, as shown by the **efskeymgr -D** and **efskeymgr -v** commands.

- ► file1, file2, and file3 are removed again.

- ► user6 restores successfully all files using the **restore** command.

- ► Any new generated shell will not contain key2, as shown by the **efskeymgr -o** and **efskeymgr -V** commands.

- ► user6 can access file3 using the active key (key3) and file1 using the deprecated key (key1). file2 can no longer be accessed because key2 has been deleted.

Example 2-48   Access keys must be saved along with files

```
$ echo 1111 > file1
$ efsmgr -l file1
EFS File information:
 Algorithm: AES_128_CBC
List of keys that can open the file:
 Key #1:
  Algorithm       : RSA_1024
  Who             : uid 208
  Key fingerprint : c566b5cf:04921ffa:1480a2bc:b34d48f3:c9cccb9c
$ efskeymgr -v
 Keystore content:
     Keystore owner ............ : uid 208
     Keystore mode ............. : admin: managed by EFS administrator
     Password changed last on .. : 05/11/07 at 16:10:19
   Private key:
         Algorithm : RSA_1024
       Fingerprint : c566b5cf:04921ffa:1480a2bc:b34d48f3:c9cccb9c
          Validity : This key is valid.


$ efskeymgr -R RSA_1024
```

```
$ efskeymgr -o ksh
user6's EFS password:
$ touch file2
$ echo 2222> file2
EFS File information:
 Algorithm: AES_128_CBC
List of keys that can open the file:
 Key #1:
  Algorithm       : RSA_1024
  Who             : uid 208
  Key fingerprint : ffcbef78:cf865d70:bfab5101:ab572d70:f3326d79

$ efskeymgr -R RSA_1024
$ efskeymgr -o ksh
user6's EFS password:
$ echo 3333> file3
$ efsmgr -l file3
EFS File information:
 Algorithm: AES_128_CBC
List of keys that can open the file:
 Key #1:
  Algorithm       : RSA_1024
  Who             : uid 208
  Key fingerprint : 340e3449:a1fb5712:6163de83:02c08db1:400d830e
$ efskeymgr -v
 Keystore content:
     Keystore owner ............ : uid 208
     Keystore mode ............. : admin: managed by EFS administrator
     Password changed last on .. : 05/11/07 at 16:10:19
   Private key:
         Algorithm : RSA_1024
       Fingerprint : 340e3449:a1fb5712:6163de83:02c08db1:400d830e
          Validity : This key is valid.
   Private key:
         Algorithm : RSA_1024
       Fingerprint : ffcbef78:cf865d70:bfab5101:ab572d70:f3326d79
          Validity : This key was deprecated on 05/11/07 at 16:41:27.
   Private key:
         Algorithm : RSA_1024
       Fingerprint : c566b5cf:04921ffa:1480a2bc:b34d48f3:c9cccb9c
          Validity : This key was deprecated on 05/11/07 at 16:33:10.
$ ls file*|backup -ivZf archive
Mount volume 1 on archive.
        Press Enter to continue.
Backing up to archive.
```

```
Cluster 51200 bytes (100 blocks).
Volume 1 on archive
a        4096 file1
a        4096 file2
a        4096 file3
The total size is 12288 bytes.
Backup finished on Fri May 11 17:03:46 CDT 2007; there are 100 blocks
on 1 volumes.
$ rm file*
$ restore -xvf archive
Please mount volume 1 on archive.
        Press the Enter key to continue.


New volume on archive:
Cluster size is 51200 bytes (100 blocks).
The volume number is 1.
The backup date is: Fri May 11 17:03:45 CDT 2007
Files are backed up by name.
The user is root.
x        4096 file1
x        4096 file2
x        4096 file3
The total size is 12288 bytes.
The number of restored files is 3.
$ efsmgr -l file1
EFS File information:
 Algorithm: AES_128_CBC
List of keys that can open the file:
 Key #1:
  Algorithm      : RSA_1024
  Who            : uid 208
  Key fingerprint : c566b5cf:04921ffa:1480a2bc:b34d48f3:c9cccb9c
$ efsmgr -l file2
EFS File information:
 Algorithm: AES_128_CBC
List of keys that can open the file:
 Key #1:
  Algorithm      : RSA_1024
  Who            : uid 208
  Key fingerprint : ffcbef78:cf865d70:bfab5101:ab572d70:f3326d79
$ efsmgr -l file3
EFS File information:
 Algorithm: AES_128_CBC
List of keys that can open the file:
 Key #1:
```

```
   Algorithm       : RSA_1024
   Who             : uid 208
   Key fingerprint : 340e3449:a1fb5712:6163de83:02c08db1:400d830e
$ cat file*
1111
2222
3333
$ efskeymgr -D ffcbef78:cf865d70:bfab5101:ab572d70:f3326d79
$ efskeymgr -v
 Keystore content:
      Keystore owner ............ : uid 208
      Keystore mode ............. : admin: managed by EFS administrator
      Password changed last on .. : 05/11/07 at 16:10:19
    Private key:
          Algorithm : RSA_1024
        Fingerprint : 340e3449:a1fb5712:6163de83:02c08db1:400d830e
           Validity : This key is valid.
    Private key:
          Algorithm : RSA_1024
        Fingerprint : c566b5cf:04921ffa:1480a2bc:b34d48f3:c9cccb9c
           Validity : This key was deprecated on 05/11/07 at 16:33:10.
$ rm file*
$ restore -xvf archive
Please mount volume 1 on archive.
        Press the Enter key to continue.


New volume on archive:
Cluster size is 51200 bytes (100 blocks).
The volume number is 1.
The backup date is: Fri May 11 17:03:45 CDT 2007
Files are backed up by name.
The user is root.
x        4096 file1
x        4096 file2
x        4096 file3
The total size is 12288 bytes.
The number of restored files is 3.
$ efskeymgr -o ksh
$ cat file*
1111
cat: 0652-050 Cannot open file2.
3333
```

> **Important:** If you are using a software product for backup, make sure that all user and group keystores are also backed up.

> **Tip:** Users may accidentally delete deprecated keys or company security policy may enforce that user deprecated keys should be deleted. To avoid losing information in such circumstances, you could update file security information by simply reencrypting user files. The newly-generated symmetric key will be encrypted with the user active public key and added to file cryptographic metadata, as shown in Example 2-49.

Example 2-49   Reencrypting files with active user keys

```
$ efskeymgr -v
 Keystore content:
     Keystore owner ............ : uid 208
     Keystore mode ............. : admin: managed by EFS administrator
     Password changed last on .. : 05/11/07 at 16:10:19
    Private key:
         Algorithm : RSA_1024
       Fingerprint : 340e3449:a1fb5712:6163de83:02c08db1:400d830e
          Validity : This key is valid.
    Private key:
         Algorithm : RSA_1024
       Fingerprint : c566b5cf:04921ffa:1480a2bc:b34d48f3:c9cccb9c
          Validity : This key was deprecated on 05/11/07 at 16:33:10.
$ efsmgr -l file1
EFS File information:
 Algorithm: AES_128_CBC
List of keys that can open the file:
 Key #1:
  Algorithm       : RSA_1024
  Who             : uid 208
  Key fingerprint : c566b5cf:04921ffa:1480a2bc:b34d48f3:c9cccb9c
$ efsmgr -e file1
$ efsmgr -l file1
EFS File information:
 Algorithm: AES_128_CBC
List of keys that can open the file:
 Key #1:
  Algorithm       : RSA_1024
  Who             : uid 208
  Key fingerprint : 340e3449:a1fb5712:6163de83:02c08db1:400d830e
```

3

Role Based Access Control

In this chapter, we introduce and discuss the use of Role Based Access Control.

This chapter contains the following sections:

3.1 AIX V6 and Role Based Access Control (RBAC)

AIX V6 introduces Enhanced RBAC, which is a method to delegate roles and authorizations among one or more general user accounts.

RBAC provides the system administrator with the ability to designate tasks to general users that traditionally would be performed by the root user, or through setuid/setgid.

One benefit of RBAC is the minimizing of the use of the setuid/setgid programs by restricting the privileges assigned to a command to only those privileges that the command needs to execute its task.

There is no specific install package in AIX V6 for either Legacy or Enhanced Mode RBAC. The majority of the Enhanced RBAC commands are included in the bos.rte.security fileset.

The following sections will introduce and further discuss the components that are included with Enhanced RBAC.

3.2 The traditional approach to AIX administration

Here we talk about the traditional approach to AIX administration and the tools used for it.

3.2.1 The superuser administrative account

The traditional approach to privileged administration in the AIX operating system has relied on a single system administrator account named *root*. The root account is classed as the superuser, as the root user account has the authority to perform all privileged system administration on the AIX system. The root user is normally designated with user identity/uid 0.

Reliance on a single superuser for all aspects of system administration poses problems in regards to the separation of administrative duties. While a single administrative account may be acceptable in certain business environments, many environments require multiple administrators, each with responsibility for performing different tasks.

Using a single administrative account may require that the superuser role is shared among two or more system administrators. This shared administrative approach may breach business audit guidelines in an environment that requires that all privileged system administration is attributable to a single individual.

An alternative method of sharing the superuser role relied on creating another user that has the same UID as the root user.

Sharing administration in either of these fashions may create issues from a security perspective in that each administrator is granted complete control over the system. There was no way to limit the operations that any given administrator could perform. Since the root user is the most privileged user, the user could perform unauthorized operations and also be able to erase any audits of these activities, thereby making tracking of the administrative actions impossible.

3.2.2 Discretionary Access Control (DAC)

Discretionary Access Controls (DACs) are the aspects of security that are under the control of the owner of a file or directory.

In AIX, DAC is provided using the traditional file object permission bit method of owner/group/other and read/write/execute.

By using file object permission bits, an individual user determines whether another user or group needs access to the data in a particular file object. The DAC is generally based on the *need to know* criterion and grants or denies access accordingly. This type of access is based on the UID and the GID(s) to which a user belongs. All file system objects have associated permissions to describe access for the owner, group, and other.

> **Attention:** Using DAC a single user ID can be designated as the owner of an executable file where only they can run it. If that user leaves the company, then a systems administrator would have to modify the DAC on that file before anyone else could run it. The same scenario would apply when using DAC with group IDs too.

In Example 3-1, we see the file objects contained in the oper1 user home directory. The file MyBankBallance.txt is an example of how DAC may be used to restrict the access to a file to an individual user or group.

Example 3-1   File object permission bit DAC for user "oper1"

```
oper1@trinity:/home/oper1# ls -ltra
total 48
-rwxr-----    1 oper1    staff            254 Apr 18 07:34 .profile
drwxr-xr-x   10 bin      bin              256 Apr 18 07:35 ..
-rw-r--r--    1 oper1    staff            553 Apr 19 23:24 smit.transaction
-rw-r--r--    1 oper1    staff            418 Apr 19 23:24 smit.script
-rw-r--r--    1 oper1    staff           1079 Apr 19 23:24 smit.log
drwxr-xr-x    2 oper1    staff            256 Apr 20 01:28 .
-rw-------    1 oper1    staff             79 Apr 20 01:59 MyBankBallance.txt
-rw-------    1 oper1    staff            390 Apr 20 02:00 .sh_history
oper1@trinity:/home/oper1#
```

3.2.3 Authorization with User ID (UID) and Group ID (GID)

As we discussed previously, DAC provides the ability to restrict access to a file object based on read, write, or execute permissions. While DAC allows some granularity by using the AIX file permissions and ownerships, DAC is unable to protect against the malicious or accidental modification of a file object's permission or ownerships. If a file object is an executable program, then using UID/GID access control is one method of further restricting access to only users with the appropriate UID/GID.

If UID/GID checking is included in an executable program, then the program will only successfully execute if the process UID/GID matches the embedded UID/GID that has been included in the executable.

In the following example, we demonstrate:

1. Modifying a privileged file object's DAC settings to allow execution by all users

2. Attempting to execute the `shutdown` command by a user that is not the root user or in the shutdown group

The `shutdown` command is shell script owned by the root user and includes read and execute permissions for the shutdown group. The shutdown shell script includes a UID/GID checking mechanism to check whether the UID/GID is root:shutdown. If the UID/GID does not comply, then the shutdown shell script will call the `exec_shutdown` executable.

To allow all users on the system to execute the **shutdown** command, we must modify the permissions to allow read and execute for users outside of the shutdown group.

In Example 3-2, we modify the **shutdown** and **exec_shutdown** commands to allow execution by all users on the system.

Example 3-2   Modifying the DAC permissions to allow execution

```
root@trinity:/root# ls -ltra /usr/sbin/shutdown
-r-xr-x---   1 root     shutdown 42939 Apr 17 10:26 /usr/sbin/shutdown
root@trinity:/root# ls -ltra /usr/sbin/exec_shutdown
-r-xr-x---   1 root     shutdown 2694 Apr 17 10:26 /usr/sbin/exec_shutdown
root@trinity:/root# chmod 555 /usr/sbin/shutdown
root@trinity:/root# chmod 555 /usr/sbin/exec_shutdown
root@trinity:/root# ls -ltr /usr/sbin/shutdown
-r-xr-xr-x   1 root     shutdown 42939 Apr 17 10:26 /usr/sbin/shutdown
root@trinity:/root# ls -ltra /usr/sbin/exec_shutdown
-r-xr-xr-x   1 root     shutdown 2694 Apr 17 10:26 /usr/sbin/exec_shutdown
root@trinity:/root#
```

Now that we have modified the DAC permissions of the **shutdown** and **exec_shutdown** commands to allow execution by any user on the system, we will log in as the user oper1 and execute the **shutdown** command. The user oper1 has a UID of 208 and is in the staff group.

In Example 3-3, we execute the **shutdown** command as the user oper1.

Example 3-3   Executing the shutdown command as user oper1

```
oper1@trinity:/home/oper1# id
uid=208(oper1) gid=1(staff)
oper1@trinity:/home/oper1# ls -ltra /usr/sbin/shutdown
-r-xr-xr-x   1 root     shutdown 42939 Apr 17 10:26 /usr/sbin/shutdown
oper1@trinity:/home/oper1# ls -ltra /usr/sbin/exec_shutdown
-r-xr-xr-x   1 root     shutdown 2694 Apr 17 10:26 /usr/sbin/exec_shutdown
oper1@trinity:/home/oper1# shutdown -F
oper1@trinity:/home/oper1#
```

The **shutdown** command executes, but rather than performing the system shutdown task, the command exits without the expected system shutdown action being performed. This is because the **exec_shutdown** executable also checks for the UID/GID of root:shutdown. If the UID/GID do not match the values coded into the **exec_shutdown** executable, then the program will exit.

This example of UID/GID access checking is not widely used in AIX 5L. Not all critical commands have been coded this way to ensure that, regardless of the DAC setting, the command will only perform the action when executed by a privileged UID/GID.

In Figure 3-1, we outline the UID/GID process used by the `shutdown` command to determine whether the oper1 user has the appropriate authorization to execute the command and perform the shutdown procedure.
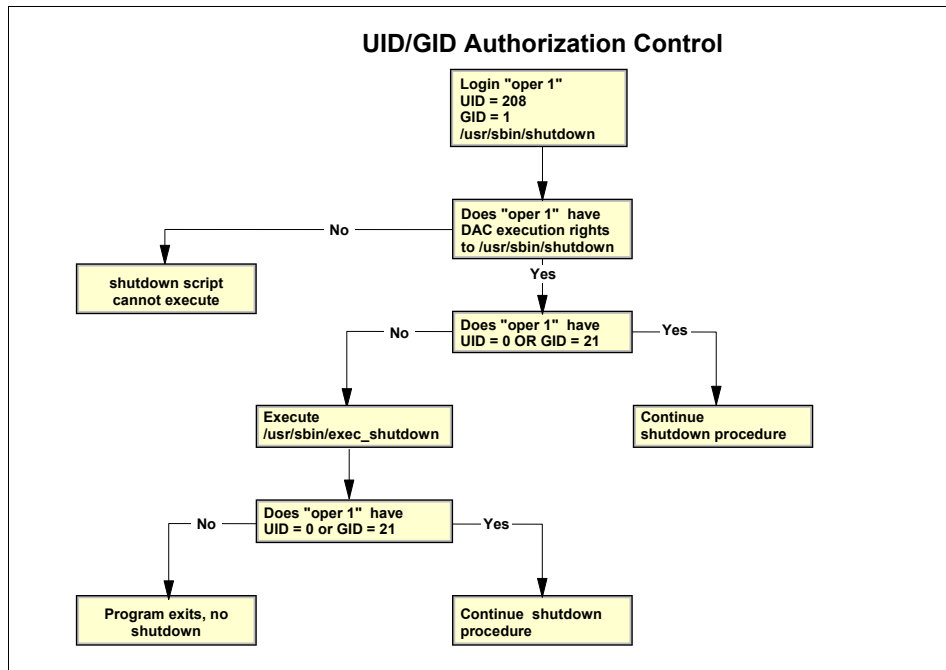

Figure 3-1   Explanation of UID/GID authorization control

3.2.4 Privileged escalation with Set User Identification (setuid)

The traditional approach to access control in AIX has been performed by using the user identity associated with the process to determine access control based on the DAC permissions of the executable. However, the root user ID of 0 has traditionally been allowed to bypass permission checks, so a process that is executing as the root user is able to succeed at any access checks on the system and perform any operation desired. This becomes a security issue when using setuid applications.

The concept of setuid allows for a command to execute under a different user identity then the user that invoked the command. setuid may be necessary for situations when a general user needs to accomplish a privileged task. An example of this is the `passwd` command. The /etc/passwd file uses DAC permissions that restrict the read/write access to the file to the root user. Since any user other then the root user not have access to the file that stores user passwords, a general user needs additional privilege to change their password. For this reason, the `passwd` command is setuid to the root user. When a user other than the root user invokes the `passwd` command, it will appear to the operating system that the root user is accessing the file and the access will be granted.

While the setuid concept does allow for the desired access control functionality, it carries with it an inherent security risk. Since the setuid program is effectively executing in the context of the root user, if an attacker successfully takes over the program that is being executed with the setuid bit before the program exits, then the attacker would have all of the powers of the root user. The attacker may then be able to bypass all access checks, as the attacker may remain with the effective privileges of the root user and be able to perform all operations that the root user could perform.

A more secure solution to privileged command execution would be to only assign a subset of the root user privileges to the program so that the least privileged principle is followed and the threat is mitigated.

The least privileged principle is an approach to security that grants a process or an individual only the authority or privilege required to perform a specific task. The user's access to additional files or commands are restricted on a need to know basis.

3.3 Introducing RBAC

In this section, we will introduce the components included in RBAC.

3.3.1 Legacy Mode versus Enhanced Mode RBAC

A limited implementation of RBAC was introduced in AIX V4.2. Beginning with AIX Version 6.1, a new implementation of RBAC provides for a very fine granular mechanism through which administration tasks can be better controlled, offering the administrator a more precise and customized approach than control of the privileged command execution.

Since the two implementations of RBAC offer significant differences in their operational scope and functionality, the two RBAC implementations will use the following terms to describe their associated implementation:

Legacy RBAC Mode The historic behavior of AIX roles introduced in AIX V 4.2.1

Enhanced RBAC Mode
The new implementation introduced with AIX Version 6.1

Both modes of operation are supported in AIX V6, however, Enhanced RBAC Mode will be the default on a newly installed system.

Legacy Mode RBAC

With the release of AIX V4.2.1, the AIX security infrastructure began to provide limited RBAC functionality. This functionality, now know as Legacy Mode RBAC, could allow users other than the root user to perform certain privileged system administration tasks. In an Legacy RBAC implementation, when a given administrative command is invoked by a user other than the root user, the code in the executed command will determine if the user was assigned a role with the required authorization. If the user is authorized, then the execution continues; if not, the command fails with an error. Legacy RBAC often requires that the command being controlled by an authorization be setuid to the root user in order for an authorized invoker to have the proper privileges to accomplish the operation.

The Legacy RBAC implementation also introduced a predefined set of authorizations that can be used to determine access to administrative commands. The predefined authorizations could be expanded by the administrator. Additionally, a framework of administrative commands and interfaces to create roles, assign authorizations to roles, and assign roles to users is also provided.

While the Legacy RBAC implementation provides the ability to partially segment the administration responsibilities, it includes the following restrictions:

► Framework requires changes to commands/applications for them to be RBAC enabled.

► Predefined authorizations are not granular as Enhanced Mode RBAC.

► Users often required membership in a certain group as well as having a role with a given authorization in order to execute a command.

► A true separation of duties is difficult to implement. If a user is assigned multiple roles, then all assigned roles are always active. There is no method to activate only a single role without activating all roles that the user is assigned.

► The least privilege principle not adopted in the operating system. Privileged commands must typically be setuid to the root user.

While Legacy RBAC will continue to be supported, administrators are strongly encouraged to move to Enhanced RBAC. Enhanced RBAC offers more granular control of authorizations and its reduced reliance upon setuid programs.

Enhanced Mode RBAC

Beginning with AIX V6, a further implementation of RBAC has been provided. This Enhanced Mode RBAC allows applications that require administrative privileges for certain operations to be integrated into new functions included with the Enhanced RBAC infrastructure.

The Enhanced RBAC integration options use granular privileges and authorizations and allow the administrator the ability to configure any command on the system as a privileged command. Features of Enhanced RBAC will be installed and enabled by default on all installations of AIX beginning with AIX V6.

Enhanced RBAC allows the administrator to provide for a customizable set of authorizations, roles, privileged commands, devices, and files through the Enhanced RBAC security databases.

With enhanced RBAC, the security databases may reside either in the local file system or be managed remotely through LDAP.

Enhanced RBAC consists of the following security database files:

► Authorization Database

► Role Database

► Privileged Command Database

► Privileged Device Database

► Privileged File Database

The Enhanced RBAC security database files are text files and do not require an additional database subsystem to be installed on the AIX V6 system.

The Enhanced RBAC databases are further discussed later in this chapter.

Enhanced RBAC mode introduces a new naming convention for authorizations that allows for a hierarchy of authorizations to be created. Enhanced RBAC includes a granular set of system-defined authorizations and allows an administrator to create additional user-defined authorizations as necessary.

Note: At the time of publication, Enhanced RBAC may be managed through the command line or the SMIT tool. Enhanced RBAC support is not included in WebSM.

3.3.2 Authorizations

In Enhanced RBAC, an authorization is a text string associated with security related functions or commands. Authorizations provide a mechanism to grant rights to users to perform privileged actions and to provide different levels of functionality to different classes of users. Authorizations are assigned to roles, which may then be assigned to user.

When a command that is governed by an authorization is executed, access is only granted if the invoking user has the required authorization. For this reason, an authorization can be thought of as a key that is able to unlock one or more commands.

3.3.3 Roles

With Enhanced RBAC, the behavior of roles has been further developed to provide for a separation of duty functionality. Enhanced RBAC introduces the concept of *role sessions* to AIX. A role session is defined as a process that has one or more roles associated to it. Enhanced RBAC allows a user to choose to activate a role session for any roles that they have been assigned. By default, none of the users roles are active at login, giving the user the ability to activate the role that is required for the currently activity.

Roles have further been enhanced to support the requirement that the user must authenticate before activating the role. This authentication requires the user's login password to be authenticated before one of the user's assigned roles may be activated.

This authorization enhancement aids in protection of an attacker taking over a user session since the attacker would still need to then authenticate by entering the user's login password to activate the user's roles.

3.3.4 Privileges

The introduction of the Privileged Command Database allows for the implementation of the *least privileged principle*. The least privileged principle is a methodology that aims to assign a user only the minimum required privileges for the user to complete a task/command/process.

Enhanced RBAC increases the granularity of privileges on the system allowing for explicit privileges to be granted to a command and the execution of that command to be governed by an authorization.

The Privileged Command Database provides the ability to remove the dependency on setuid and setgid programs, allows the administrator to assign commands only the privileges that are required for the successful execution of the command, without requiring a code change to the actual command.

The Privileged Device Database allows read and write access to devices to be controlled by privileges.

The Privileged File Database will allow unprivileged users read and write access to restricted files based on authorizations.

The privileged command database, privileged command, and privileged file database increase the granularity of system administrative tasks that can be assigned to otherwise non-privileged users.

3.3.5 Kernel Security Tables

Enhanced RBAC provides a mechanism whereby information in the Enhanced RBAC security database is gathered and verified and then sent to an area of the kernel designated as the Kernel Security Tables (KST). The data in the KST determines the security policy for the system. Entries modified in the file or LDAP (also known as the *user-level*) RBAC security database are not used for security decisions until that information has been verified and updated into the KST (also known as the *kernel-level*).

The KST is updated with the `setkst` command. The contents of the KST may be displayed with the `lskst` command.

3.3.6 Remote database support using LDAP

In an environment that includes many servers, it may be more efficient to maintain the RBAC security database in a centralized location. This may also be true in an enterprise environment where implementing and enforcing a common security policy across all systems is desired.

When the RBAC security databases that control the security policy are stored independently on each system, management of the security policy becomes a burden on the designated administrator.

The Enhanced RBAC mode in AIX V6 allows for the Enhanced RBAC security database to be stored in LDAP. This allows the security policy to be centrally managed for all systems in the LDAP environment.

Support is included in AIX V6 for all of the Enhanced RBAC security databases to be located in LDAP, including:

► Authorization Database

► Role Database

► Privileged Command Database

► Privileged Device Database

► Privileged File Database

> **Restriction:** Authorization databases stored in LDAP will only contain the user-defined authorizations. System-defined authorizations cannot be stored in LDAP and will remain local to each client system.

Included in AIX V6 are utilities that allow the administrator to:

► Export local RBAC security database data to LDAP.

► Configure the AIX V6 client to make use of RBAC data in LDAP.

► Control the domain lookup of RBAC security database data.

► Manage the RBAC security database LDAP data from a client system.

Remote database support using LDAP will be discussed in more detail in 3.14, "RBAC remote database support" on page 228.

3.3.7 Legacy and Enhanced RBAC mode comparison

Table 3-1 compares the features available in each mode of RBAC.

Table 3-1   RBAC Legacy and Enhanced Mode features

Feature	Legacy RBAC mode	Enhanced RBAC mode
Selective role activation	All user roles are active by default.	No roles are active by default. Roles are assumed with the `swrole` command.
default_roles attribute	No.	Yes.
`swrole` command	No.	Yes.
Role management commands	Yes.	Yes.
Authorization management commands	Yes.	Yes.
Authorization Hierachy	No authorization hierarchy, each authorization is independent.	Supports the concept of authorization hierarchy.
Authorization checking	Only enforced if the command checks for authorization at execution.	Enforced through the Privileged Command Database or by command check authorization at execution.
Granular privileges	Yes.	Yes.
`pvi` command	No.	Yes.
Kernel Security Tables	No.	Yes.
RBAC database location	Local files.	Local files or LDAP.

Table 3-2 shows the sizing limitations for Enhanced RBAC.

Table 3-2   Sizing limitations for Enhanced RBAC

Description	Limitation
Maximum Role name size	63 printable characters
Maximum roles per session	8
Maximum Authorization name size	63 printable characters
Maximum number of levels in an Authorization hierarchy (including the top level parent)	9

Description	Limitation
Maximum number of access authorizations per command	8
Maximum Authorized Privileged Sets per command	8

3.4 Configuring RBAC

In this section, we will outline the installation and configuration of RBAC in AIX V6.

3.4.1 Configuring the RBAC operating mode

As discussed in 3.3, "Introducing RBAC" on page 171, when AIX V6 is installed, RBAC is, by default, activated in Enhanced Mode.

To determine the current mode in which RBAC is operating, display the enhanced_RBAC mode of the sys0 resource.

In Example 3-4, we display the currently active RBAC mode using the `lsattr` command and the sys0 resource.

Example 3-4   Displaying the RBAC mode

```
root@trinity:/root# lsattr -El sys0 -a enhanced_RBAC
enhanced_RBAC true Enhanced RBAC Mode True
root@trinity:/root#
```

3.4.2 Switching to Legacy RBAC mode

If you want to switch to Legacy RBAC mode, then this can be accomplished by setting the enhanced_RBAC attribute to `false`. Changing the RBAC mode requires a reboot of the server.

In Example 3-5, we use the **chdev** command to change the RBAC mode from Enhanced to Legacy.

Example 3-5   Changing the RBAC mode from Enhanced to Legacy

```
root@trinity:/root# lsattr -El sys0 -a enhanced_RBAC
enhanced_RBAC true Enhanced RBAC Mode True
root@trinity:/root# chdev -l sys0 -a enhanced_RBAC=false
sys0 changed
root@trinity:/root# lsattr -El sys0 -a enhanced_RBAC
enhanced_RBAC false Enhanced RBAC Mode True
root@trinity:/root# shutdown -Fr
```

Note: In a WPAR environment, the RBAC mode for the system will only be configurable from the global system and will uniformly affect the global as well as all the WPARs on the system.

3.4.3 The root user and Enhanced RBAC

When using RBAC in Enhanced RBAC mode, the root user will continue to function as in prior releases of AIX 5L and can remain in its traditional role as the superuser account.

A feature of Enhanced RBAC is the ability to disable the root user account and perform all privileged administration and commands through one or more user accounts.

Disabling the root user account in Enhanced RBAC mode will be further discussed in 3.11, "Securing the root user" on page 219.

3.5 Predefined roles in RBAC

In this section, we will discuss adding a predefined role to a user.

With the release of Enhanced RBAC in AIX V6, three predefined roles and several subroles are included with the default configuration of RBAC.

The three predefined roles are:

► ISSO: Information System Security Officer

The ISSO role is responsible for creating and assigning roles and is thus the most powerful user-defined role on the system. Some of the ISSO responsibilities include:

– Establishing and maintaining security policy

– Setting passwords for users

– Network configuration

– Device administration

► SA: Systems Administrator

The SA role provides the functionality for daily administration and is responsible for:

– User administration (except password setting)

– File system administration

– Software installation update

– Network daemon management

– Device allocation

► SO: System Operator

The SO role provides the functionality for day to day operations and is responsible for:

– System shutdown and reboot

– File system backup, restore, and quotas

– System error logging, trace, and statistics

– Workload administration

Each of these predefined roles come pre-configured with authorization definitions and may be further customized, if required.

Note: The ISSO, SA, and SO roles are used by Trusted AIX. If your environment includes Trusted AIX, you may wish to consider customizing your RBAC environment by using user defined roles.

3.5.1 Adding a role to a user

As discussed earlier in this section, the *so* predefined role includes the authorization to execute the **reboot** and **shutdown** commands.

If we did not know if a predefined role included the **shutdown** and **reboot** commands, the procedure to follow would be:

1. Identify the privileged command that needs to be executed. In this case, the the **shutdown** and **reboot** commands need to be executed.

   Add the fully qualified privileged command to the entry field area of the smitty menu and select the Enter key.

   Figure 3-2 shows the **smit setsecattr_cmdmod** fast path.

```
                 Change/Show Characteristics of a Privileged Command

Type or select a value for the entry field.
Press Enter AFTER making all desired changes.

                                                         [Entry Fields]
* Command Name                                       [/usr/sbin/exec_shutdown]   +













F1=Help              F2=Refresh            F3=Cancel            F4=List
F5=Reset             F6=Command            F7=Edit              F8=Image
F9=Shell             F10=Exit              Enter=Do
```

Figure 3-2   The smitty setsecattr_cmdmod fast path

2. Determine if the privileged command is included in an authorization. If the command is not included in a predefined authorization, then a user-defined authorization may need to be defined.

   The **smit setsecattr_cmdmod** command displays the authorization for the **/usr/sbin/exec_shutdown** command as `aix.system.boot.shutdown`.

   Using the same method, we determine that the **/usr/sbin/reboot** command is defined in the `aix.system.boot.reboot` authorization.

   Alternatively, the **lssecattr** command could be used to obtain the information without invoking the smitty menu.

Figure 3-3 shows the `lssecattr` command.

```
root@trinity:/root# lssecattr -c /usr/sbin/exec_shutdown
/usr/sbin/exec_shutdown accessauths=aix.system.boot.shutdown
innateprivs=PV_DAC_R secflags=FSF_EPS

root@trinity:/root# lssecattr -c /usr/sbin/reboot
/usr/sbin/reboot accessauths=aix.system.boot.reboot
innateprivs=PV_KER_REBOOT,PV_SU_UID secflags=FSF_EPS
root@trinity:/root#
```

Figure 3-3   lssecattr command

Figure 3-4 shows the `exec_shutdown` command with `smitty setsecattr_cmdmod`.

```
                    Change/Show Characteristics of a Privileged Comamnd

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

[TOP]                                                [Entry Fields]
* Command Name                                        /usr/sbin/exec_shutdown
  Effective User ID                                  [1]                        +#
  Effective Group ID                                 []                         +#
  Access Authorizations                              [aix.system.boot.shutdown] +
  Innate Privileges                                  [PV_DAC_R]                 +
  Inherited Privileges                               []                         +
  Security Flags                                     [FSF_EPS]                  +

  Authorized Privileges
     (Up to 8 Authorizations can be specified.
      Privileges must be specified for
      authorizations selected)

[MORE...16]

F1=Help            F2=Refresh          F3=Cancel           F4=List
F5=Reset           F6=Command          F7=Edit             F8=Image
F9=Shell           F10=Exit            Enter=Do
```

Figure 3-4   exec_shutdown command with smitty setsecattr_cmdmod

3. Determine if there is an appropriate role that includes the authorization. If an appropriate role is not included in the predefined roles, then a user-defined role may need to be defined.

   The `lsrole` command may be used to list the defined roles. The `lsrole` and `grep` commands can be combined to determine if an authorization is assigned to a role.

Figure 3-5 shows the *so* role contains several authorizations, including `aix.system.boot.reboot` and `aix.system.boot.shutdown`.

```
root@trinity:/root# lsrole ALL | grep -i aix.system.boot.shutdown
so
authorizations=aix.fs.manage.backup,aix.fs.manage.restore,aix.proc.k
ill,aix.ras.debug,aix.ras.dump,aix.ras.error,aix.ras.trace,aix.syste
m.boot.halt,aix.system.boot.info,aix.system.boot.reboot,aix.system.b
oot.shutdown,aix.system.config.init,aix.system.config.wlm rolelist=
groups= visibility=1 screens=* msgcat= id=3
root@trinity:/root#
```

Figure 3-5   The lsrole command

Another option would be to use the **lsauth** command with the roles attribute to display the role that an authorization is assigned to.

The **lsauth** command may be used with the " * " wildcard. The wild card may be used at the end of a name to list an entire hierarchy. The entire string specified before the wild card must be a valid authorization name.

Figure 3-6 shows the **lsauth** command used with the roles attribute.

```
root@trinity:/usr/sbin# lsauth -a roles aix.system.boot.*
aix.system.boot roles=
aix.system.boot.create roles=isso
aix.system.boot.halt roles=so
aix.system.boot.info roles=so
aix.system.boot.reboot roles=so
aix.system.boot.shutdown roles=so
root@trinity:/usr/sbin#
```

Figure 3-6   The lsauth command

By using a combination of the **lsrole** and **lsauth** commands, we can determine the role or roles that an authorization is assigned to as well as the authorizations that are assigned to a role.

4. Assign the role to the user.

    RBAC roles may be assigned with the `chuser` command.

    Figure 3-7 shows the `lsuser` command output prior to the so role being
    assigned to the oper1 user. The oper1 user currently has no assigned roles,
    therefore the roles value is blank.

```
root@trinity:/usr/sbin# lsuser oper1
oper1 id=208 pgrp=staff groups=staff home=/home/oper1
shell=/usr/bin/ksh gecos=operator login=true su=true rlogin=true
daemon=true admin=false sugroups=ALL admgroups= tpath=nosak ttys=ALL
expires=0 auth1=SYSTEM auth2=NONE umask=22 registry=files
SYSTEM=compat logintimes= loginretries=5 pwdwarntime=0
account_locked=false minage=1 maxage=13 maxexpired=-1 minalpha=1
minother=1 mindiff=1 maxrepeats=2 minlen=6 histexpire=4 histsize=4
pwdchecks= dictionlist= default_roles= fsize=2097151 cpu=-1
data=262144 stack=65536 core=2097151 rss=65536 nofiles=2000
time_last_login=1177326441 tty_last_login=/dev/pts/1
host_last_login=9.41.222.87 unsuccessful_login_count=0 roles=
```

Figure 3-7   lsuser of oper1 with no role assigned

    The `chuser` command does not append new roles to existing roles. If the
    oper1 user had existing roles assigned, these existing roles would need to be
    included in the new role value in the `chuser` command.

Figure 3-8 shows adding the so role to the oper1 user with the `chuser` command.

```
root@trinity:/usr/sbin# chuser roles=so oper1
root@trinity:/usr/sbin# lsuser oper1
oper1 id=208 pgrp=staff groups=staff home=/home/oper1
shell=/usr/bin/ksh gecos=operator login=true su=true rlogin=true
daemon=true admin=false sugroups=ALL admgroups= tpath=nosak ttys=ALL
expires=0 auth1=SYSTEM auth2=NONE umask=22 registry=files
SYSTEM=compat logintimes= loginretries=5 pwdwarntime=0
account_locked=false minage=1 maxage=13 maxexpired=-1 minalpha=1
minother=1 mindiff=1 maxrepeats=2 minlen=6 histexpire=4 histsize=4
pwdchecks= dictionlist= default_roles= fsize=2097151 cpu=-1
data=262144 stack=65536 core=2097151 rss=65536 nofiles=2000
time_last_login=1177326441 tty_last_login=/dev/pts/1
host_last_login=9.41.222.87 unsuccessful_login_count=0 roles=so
root@trinity:/usr/sbin#
```

Figure 3-8   Assign the so role to user "oper1" with the chuser command

5. Activate the role.

   By default, a role is not active until the user logs into the system and uses the `swrole` command. If the oper1 user were to execute the `shutdown` command without activating the so role, the `shutdown` command would attempt to execute as the oper1 user. As the oper user has no special privileges until it activates one of its assigned roles, the `shutdown` command would fail to complete successfully.

   By activating the so role, the oper1 user will gain the access controls granted to the so role. Any command that is executed by the oper1 user that is included in an authorization contained in the so role will then execute with the privileges required to execute the command.

In Figure 3-9, we log in as the oper1 user and execute the `rolelist -a` command to display the roles and authorizations that oper1 is assigned.

We then execute the `rolelist -e` command to display the effective roles. The effective roles can be thought of as the roles that are currently active and available for use by the oper1 user.

As there are no effective roles active, any commands that are executed by the oper1 user will be executed without any additional privilege, so the `shutdown -Fr` command exits without performing the shutdown procedure.

```
oper1@trinity:/home/oper1# rolelist -a
so              aix.fs.manage.backup
                aix.fs.manage.restore
                aix.proc.kill
                aix.ras.debug
                aix.ras.dump
                aix.ras.error
                aix.ras.trace
                aix.system.boot.halt
                aix.system.boot.info
                aix.system.boot.reboot
                aix.system.boot.shutdown
                aix.system.config.init
                aix.system.config.wlm
oper1@trinity:/home/oper1# rolelist -e
rolelist: 1420-062 There is no active role set.
oper1@trinity:/home/oper1# shutdown -Fr
oper1@trinity:/home/oper1#
```

Figure 3-9   The rolelist command

To activate a role, use the `swrole` command.

The `swrole` command will start a new shell that will require that the user authenticate with their password.

Once the password is authenticated, the role will be active and commands that are included in the authorizations defined to the role will be executed as privileged commands.

In Figure 3-10, we execute the `swrole` command. The `swrole` command requires the role name as the argument, therefore we execute `swrole so`. This will activate the so role.

Once the so role is active, the `rolelist -ea` command can be used to list the effective role and authorizations.

```
oper1@trinity:/home/oper1# swrole so
oper1's Password:
oper1@trinity:/home/oper1# rolelist -ea
so              aix.fs.manage.backup
                aix.fs.manage.restore
                aix.proc.kill
                aix.ras.debug
                aix.ras.dump
                aix.ras.error
                aix.ras.trace
                aix.system.boot.halt
                aix.system.boot.info
                aix.system.boot.reboot
                aix.system.boot.shutdown
                aix.system.config.init
                aix.system.config.wlm
oper1@trinity:/home/oper1#
```

Figure 3-10   The swrole command

6. The oper1 user now has the so role active and may execute the `shutdown` command.

   By adding the so role to the user oper1, it allows the user to perform the shutdown and reboot procedures without access to the root user or membership to the shutdown group. Additionally, there are no file object DAC changes required.

In Figure 3-11, we execute the **shutdown** command from the oper1 user.

```
oper1@trinity:/home/oper1# id
uid=208(oper1) gid=1(staff)
oper1@trinity:/home/oper1# ls -ltra /usr/sbin/shutdown
-r-xr-x---    1 root     shutdown      42939 Apr 17 10:26
/usr/sbin/shutdown
oper1@trinity:/home/oper1# ls -ltra /usr/sbin/exec_shutdown
-r-xr-x---    1 root     shutdown       2694 Apr 17 10:26
/usr/sbin/exec_shutdown
oper1@trinity:/home/oper1# rolelist -ea
so            aix.fs.manage.backup
              aix.fs.manage.restore
              aix.proc.kill
              aix.ras.debug
              aix.ras.dump
              aix.ras.error
              aix.ras.trace
              aix.system.boot.halt
              aix.system.boot.info
              aix.system.boot.reboot
              aix.system.boot.shutdown
              aix.system.config.init
              aix.system.config.wlm
oper1@trinity:/home/oper1# shutdown -Fr

SHUTDOWN PROGRAM
Wed Apr 25 02:35:30 MST 2007
0513-044 The sshd Subsystem was requested to stop.

Wait for 'Rebooting...' before stopping.
Error reporting has stopped.
```

Figure 3-11   Execute the shutdown command from user oper1

Note: Roles should be assigned using the least privileged principle. Assigning a user a role that includes authorizations that are more than a user requires should be avoided.

3.5.2 Activating a role

By default in AIX Version 6.1 with Enhanced RBAC, the default login process does not assign or activate any of the users defined roles or associated authorizations. The user will log in with only those privileges that it obtains from DAC or, where relevant, any privileges obtained by executing setuid programs.

In order to associate a role to the session, the user must execute the `swrole` command, which will then activate one of the role(s) that the user has assigned to it. The user may only assume roles that have previously been assigned to the user.

The act of activating a role and assuming the associated authorizations is know as assuming a role-session.

By default, a user will be required to authenticate by entering their login password when entering a role session or adding a role to their session. Roles may optionally be configured to not require authentication through use of the auth_mode role attribute. Additionally, roles may be configured to automatically activate at user login through the use of the default_roles user attribute.

Role authorization and default roles will be further discussed later in this section.

Switching to a new role-session will create a new shell session without inheriting roles from the prior role-session. This is accomplished by creating a new process shell for the role-session and assigning the new *role id* (rid) to the process. Creation of the new role-session is a similar procedure to that of the `su` command, except that in the role-session only the role ID of the process is changed and not the uid or gid.

The `swrole` command will allow for the user to create a role-session composed of a single role or multiple roles. A user may switch from a current role-session to a new role-session, though as the new role-session will be a new process, the new role-session will not inherit any roles from the previous role-session. In order to restore the previous role-session, the user must exit the current role-session.

Any roles assumed in a role-session or the active role set can be listed by invoking the `rolelist` command in the session. Additionally, an administrator can use the `rolelist` command to list the active role set for a given process on the system.

3.5.3 Role authentication

Roles may be defined or modified to allow the user to activate an assigned role without requiring password authentication.

By default, when a role is activated with the `swrole` command, the user will be required to authenticate by entering the user's login password.

The `chrole` command may be used to modify a role so that authentication is not required when the role is activated.

If a role has not had the auth_mode attribute modified from its default value, then no value will be displayed for the auth_mode attribute or stanza.

Once the auth_mode has been modified, the stanza will be displayed with the value of the auth_mode attribute.

Figure 3-12 shows the use of the `chrole` command to change the auth_mode for the so role.

```
root@trinity:/etc/security# lsrole -a auth_mode so
so
root@trinity:/etc/security# chrole auth_mode=NONE so
root@trinity:/etc/security# lsrole -a auth_mode so
so auth_mode=NONE
root@trinity:/etc/security# chrole auth_mode=INVOKER so
root@trinity:/etc/security# lsrole -a auth_mode so
so auth_mode=INVOKER
root@trinity:/etc/security#
```

Figure 3-12   changing the authmode for the so role

When the `lsrole` command is first used to display the auth_mode stanza, there is no stanza information or attribute value. This is because the auth_mode has not been modified from its default setting of INVOKER.

Once the auth_mode is set to NONE, the stanza and attribute value are displayed, and the so role would not require authentication when activated with the `swrole` command.

To reinstated the default setting, the `chrole` command is then executed to modify the auth_mode stanza back to enable INVOKER authorization. The so role will once again require authentication when activated with the `swrole` command.

Note: Modifying a role to allow its activation without authentication will allow a user to activate the role without the user entering its login credentials. This may allow an attacker to use an idle or unlocked user session to activate a role and perform restricted commands without any authentication checking.

3.5.4 Role activation

A user may have its defined role(s) optionally configured to be activated as part of the user login process by using the default_roles user attribute. The default_roles is a user attribute introduced in AIX V6 and is intended for use in situations where processes created on behalf of a user always needs to be associated with a given set of roles.

An example of a situation where the default_roles attribute would be used is the **cron** process. The **cron** process runs in the background and executes commands as the defined user. It is conceivable that some of the commands that are executed will require authorizations. This necessitates the ability to designate that a set of roles always be active for a user ID since there is no mechanism for **cron** to later acquire the role.

The default_roles user attribute can be set to include up to eight role names or the special value of ALL. Setting `default_roles=ALL` will assign all the user's roles to the session. If the user has been assigned more then eight roles, then only the first eight roles will be enabled for the session.

3.6 User defined roles

In this section, we will introduce user defined roles.

3.6.1 Planning for user defined roles

As we discussed in 3.5, "Predefined roles in RBAC" on page 179, AIX V6 includes three predefined roles. The predefined roles are a provided as a suggested means of dividing administrative duties.

The predefined roles may be modified or removed, and new roles may be created as deemed suitable for the individual environment.

> **Note:** The ISSO, SA and SO roles are used by Trusted AIX. If your environment includes Trusted AIX, you may wish to consider customizing your RBAC environment by using user defined roles.

While these predefined roles may be suitable for many administrative needs, there may be instances where a further, more granular separation of duties is required. In such instances, Enhanced RBAC allows for the creation of user defined roles.

When creating a user defined role, consideration should be given to the following points:

The name of the role The name of the role should include some description or insight into the capabilities of the role. Role names are limited to 63 printable characters.

Authorizations Consider which authorizations should be assigned to the role.

Subroles Consider whether the role should contain subroles. Subroles are a convenient way of assigning one or more pre-existing roles to a user defined role.

Authentication Should the user be required to authenticate when assuming the role through the `swrole` command.

3.6.2 Creating a user defined role

AIX V6 includes several role and KST management commands:

mkrole Create a new role. When the system is operating in enhanced RBAC mode, roles created in the role database may be immediately assigned to users but are not used for security considerations until the database has been sent to the Kernel Security Tables (KST) through the `setkst` command.

rmrole Remove an existing role. When the system is operating in enhanced RBAC mode, roles removed from the role database will still exist in the KST until the KST has been updated with the `setkst` command.

lsrole	The **lsrole** command lists the role definitions available in the roles database. When the system is operating in enhanced RBAC mode, the information in the roles database may differ from what is being used for security considerations on the system in the KST. To view the state of the roles database in the KST, use the **lskst** command.
chrole	Change or modify an existing role. When the system is operating in enhanced RBAC mode, modifications made to the role database are not used for security considerations until the database has been sent to the KST through the **setkst** command.
swrole	The **swrole** command activates a role. The activation of a role may also be referred to as "swapping" a role. Only roles that have been assigned to a user may be activated.
setkst	Updates the Enhanced RBAC Kernel Security Tables. RBAC security checking is performed at the kernel-level, so any user-level changes to the RBAC security database needs to be updated into the KST before the changes can be used for RBAC security checking.
lskst	Lists the contents of the Kernel Security Tables. The contents of the RBAC security database may not always match the KST, so the **lskst** command may be used to list or compare the KST and user-level RBAC security database.

In the previous section, we assigned the so role to the oper1. The so role allowed the oper1 user to perform the **shutdown** and **reboot** commands, but also allowed several other privileged commands, including the **kill** command.

Consider this scenario: After conducting an audit, the security controls officer has requested that we restrict the oper1 user so that it can perform only the **shutdown** and **reboot** commands and no other privileged commands. As there is no predefined role that includes only these two commands, we must create a user defined role.

To create a user defined role:

1. Log in as root or your role's administrative user.

   In Figure 3-13 on page 195, we will use the **smitty mkrole** fast path.

The following entry fields are display:

Role Name   The name that will be used for the used defined role. This user defined role will be used for shutdown and reboot operations, so we shall define the name as shutdown_reboot.

Role ID   The unique role identification number. If left blank, the next available role number will be assigned. Using the next available role number is the recommended approach.

Authorizations   The authorizations that should be assigned to the role. These may be system or user defined authorizations. In Figure 3-3 on page 182, we previously determined that the authorizations required are `aix.system.boot.shutdown` and `aix.system.boot.shutdown`. By using the F4 key, these two authorizations are chosen from the authorization list.

Role List   A list of subroles to add to this role, if required.

Groups   A list of groups to assign this role, if required.

SMIT Screens   Defines the SMITTY screens that this role will have access to. The " * " wildcard defines ALL screens.

Visibility   Specifies the role's visibility status to the system.

Description   A description of the role.

```
                              Add a Role                                  _

Type or select values in entry fields.
Press Enter AFTER making all desired changes.


                                              [Entry Fields]
* Role NAME                                 [shutdown_reboot]
  Role ID                                   []                          #
  AUTHORIZATIONS                            [aix.system.boot.reboo>    +
  Role LIST                                 []                         +
  GROUPS                                    []                         +
  Smit SCREENS                              [*]                        +
  VISIBILITY                                [1]                        +#
  Message CATALOG                           []
  Message SET                               []                          #
  Message NUMBER                            []                          #
  Description                               [Allows reboot and shu>




F1=Help            F2=Refresh         F3=Cancel          F4=List
F5=Reset           F6=Command         F7=Edit            F8=Image
F9=Shell           F10=Exit           Enter=Do
```

Figure 3-13   The smitty mkrole command

Once the field entries in the SMIT screen have been input, select Enter.

An alternative to using the SMIT screen would be to use the `mkrole` command.

Figure 3-14 shows the `mkrole` command using the same input parameters as used in Figure 3-13.

```
root@trinity:/root# mkrole dfltmsg="Allows reboot and shutdown"
authorizations="aix.system.boot.reboot,aix.system.boot.shutdown"
shutdown_reboot
root@trinity:/root#
```

Figure 3-14   the mkrole command

2. The user defined role shutdown_reboot has now been defined.

In Figure 3-15, we use the `lsrole` command to list the shutdown_reboot role. The role contains only the `aix.system.boot.reboot` and `aix.system.boot.shutdown` authorizations.

By limiting the authorizations that are available to this role, we are able to enforce the least privileged principle, granting the oper1 user access to only those commands that are required for the user's job function.

```
root@trinity:/root# lsrole shutdown_reboot
shutdown_reboot
authorizations=aix.system.boot.reboot,aix.system.boot.shutdown
rolelist= groups= visibility=1 screens=* dfltmsg=Allows reboot and
shutdown msgcat= id=4
root@trinity:/root#
```

Figure 3-15   lsrole - list a user defined role

3. Assign the role to the user.

RBAC roles may be assigned with the `chuser` command.

Figure 3-16 shows the `lsuser` command output prior to the shutdown_reboot role being assigned to the oper1 user. The oper1 user currently has the so role assigned.

```
root@trinity:/root# lsuser oper1
oper1 id=208 pgrp=staff groups=staff home=/home/oper1
shell=/usr/bin/ksh gecos=operator login=true su=true rlogin=true
daemon=true admin=false sugroups=ALL admgroups= tpath=nosak ttys=ALL
expires=0 auth1=SYSTEM auth2=NONE umask=22 registry=files
SYSTEM=compat logintimes= loginretries=5 pwdwarntime=0
account_locked=false minage=1 maxage=13 maxexpired=-1 minalpha=1
minother=1 mindiff=1 maxrepeats=2 minlen=6 histexpire=4 histsize=4
pwdchecks= dictionlist= default_roles= fsize=2097151 cpu=-1
data=262144 stack=65536 core=2097151 rss=65536 nofiles=2000
time_last_login=1177326441 tty_last_login=/dev/pts/1
host_last_login=9.41.222.87 unsuccessful_login_count=0 roles=so
root@trinity:/root#
```

Figure 3-16   lsuser of "oper1" with the so role defined

The `chuser` command does not append new roles to existing roles. If the oper1 user had existing roles assigned, these existing roles would need to be included in the new role value in the `chuser` command.

Figure 3-17 shows adding the shutdown_reboot role to the oper1 user with the `chuser` command.

```
root@trinity:/root# chuser roles=shutdown_reboot oper1
root@trinity:/root# lsuser oper1
oper1 id=208 pgrp=staff groups=staff home=/home/oper1
shell=/usr/bin/ksh gecos=operator login=true su=true rlogin=true
daemon=true admin=false sugroups=ALL admgroups= tpath=nosak ttys=ALL
expires=0 auth1=SYSTEM auth2=NONE umask=22 registry=files
SYSTEM=compat logintimes= loginretries=5 pwdwarntime=0
account_locked=false minage=1 maxage=13 maxexpired=-1 minalpha=1
minother=1 mindiff=1 maxrepeats=2 minlen=6 histexpire=4 histsize=4
pwdchecks= dictionlist= default_roles= fsize=2097151 cpu=-1
data=262144 stack=65536 core=2097151 rss=65536 nofiles=2000
time_last_login=1177326441 tty_last_login=/dev/pts/1
host_last_login=9.41.222.87 unsuccessful_login_count=0
roles=shutdown_reboot
root@trinity:/root#
```

Figure 3-17   Assign the shutdown_reboot role to user oper1 with the chuser command

4. Update the Kernel Security Tables.

When operating in Enhanced RBAC mode, authorization and privilege checks are performed in the AIX kernel. If an addition or modification is made to the RBAC role table, then this requires that the RBAC security database be updated into the AIX kernel before the modifications are available for use.

AIX V6 and Enhanced RBAC introduces the `setkst` command to update the RBAC tables into the AIX kernel.

The `setkst` command reads the RBAC security database files and loads the information from the database files into the Kernel Security Tables (KST). By default, all the security databases are sent to the KST.

Figure 3-18 shows the oper1 user executing a **rolelist** command prior to the RBAC security database files being updated into the KST with the **setkst** command.

```
oper1@trinity:/home/oper1# rolelist -e
rolelist: 1420-062 There is no active role set.
oper1@trinity:/home/oper1# swrole shutdown_reboot
swrole: 1420-050 shutdown_reboot is not a valid role.
oper1@trinity:/home/oper1# rolelist -a
rolelist: 1420-063 Role "shutdown_reboot" does not exist.
oper1@trinity:/home/oper1#
```

Figure 3-18   The rolelist command without the updated KST

Though the user defined role shutdown_reboot has been defined to the RBAC security database, the KST has not been updated with the **setkst** command since the new role has been defined. All RBAC security checks are performed against the RBAC security information located in the KST, so the security checks fail, as the shutdown_reboot role does not yet exist in the KST.

Executing the **setkst** command will update the RBAC security database into the KST.

The **setkst** will also perform a validity check of the RBAC security database tables. If an error is encounted, depending on the severity of the error, the **setkst** command will either list the error and continue or exit the procedure without updating the KST.

Figure 3-19 shows the **setkst** command. Once the **setkst** command completes successfully, the KST has been updated and the newly created role will be available for use.

```
root@trinity:/root# setkst
Successfully updated the Kernel Authorization Table.
Successfully updated the Kernel Role Table.
Successfully updated the Kernel Command Table.
Successfully updated the Kernel Device Table.
root@trinity:/root#
```

Figure 3-19   setkst command

> **Note:** A role may be assigned to a user without an update to the KST. The role will not be available to the user until updated to the KST.

5. Activate the role.

   By default, a role is not active until the user logs into the system and activates the role with the `swrole` command. If the oper1 user were to execute the `shutdown` command without activating the shutdown_reboot role, the `shutdown` command would attempt to execute as the oper1 user.

   To activate a role, we use the `swrole` command.

   The `swrole` command will start a new role-session shell that will require that the user authenticate with their password.

   Once the password is authenticated, the role will be active and commands that are included in the authorizations defined to the role will be executed as privileged commands.

   In Figure 3-20, we execute the `swrole` command. The `swrole` command requires the role name as the argument, therefore we execute `swrole shutdown_reboot`. This will activate the shutdown_reboot role.

   Once the shutdown_reboot role is active, the `rolelist -ea` command can be used to list the effective role and authorizations.

```
oper1@trinity:/home/oper1# swrole shutdown_reboot
oper1's Password:
oper1@trinity:/home/oper1# rolelist -ea
shutdown_reboot aix.system.boot.reboot
                aix.system.boot.shutdown
oper1@trinity:/home/oper1#
```

Figure 3-20   The swrole command

6. The oper1 user now has the so role active and may execute the `shutdown` and `reboot` commands.

   By adding the shutdown_reboot role to the user oper1, we can perform the shutdown and reboot procedures without access to the root user or membership to the shutdown group. Additionally, there are no file object DAC changes required.

In Figure 3-21, we execute the **shutdown** command from the oper1 user.

```
oper1@trinity:/home/oper1# id
uid=208(oper1) gid=1(staff)
oper1@trinity:/home/oper1# rolelist -ea
shutdown_reboot aix.system.boot.reboot
                aix.system.boot.shutdown
oper1@trinity:/home/oper1# shutdown -Fr

SHUTDOWN PROGRAM
Thu Apr 26 09:01:02 MST 2007
0513-044 The sshd Subsystem was requested to stop.

Wait for 'Rebooting...' before stopping.
Error reporting has stopped.
```

Figure 3-21   Execute the shutdown command from the oper1 user

3.7 System defined and user defined authorizations

In this section, we will discuss Enhanced RBAC authorizations and outline and describe the procedure to create a user defined authorization.

3.7.1 Planning for user defined authorizations

The AIX 5L implementation of RBAC provides for two type of authorizations:

System Defined   Authorizations that come predefined and are installed as a part of the AIX 5L installation process. System defined authorizations cannot be modified.

User Defined   Any authorization that is not a system defined authorization. Once created, user defined authorizations may be modified or removed.

System defined authorizations are prefixed with the "aix." name in the authorization hierarchy. System defined authorizations may not be modified or removed. User defined authorizations may not be included in the system defined authorization hierarchy. See Figure 3-22 below for an illustration of this situation.

Authorization Naming Convention
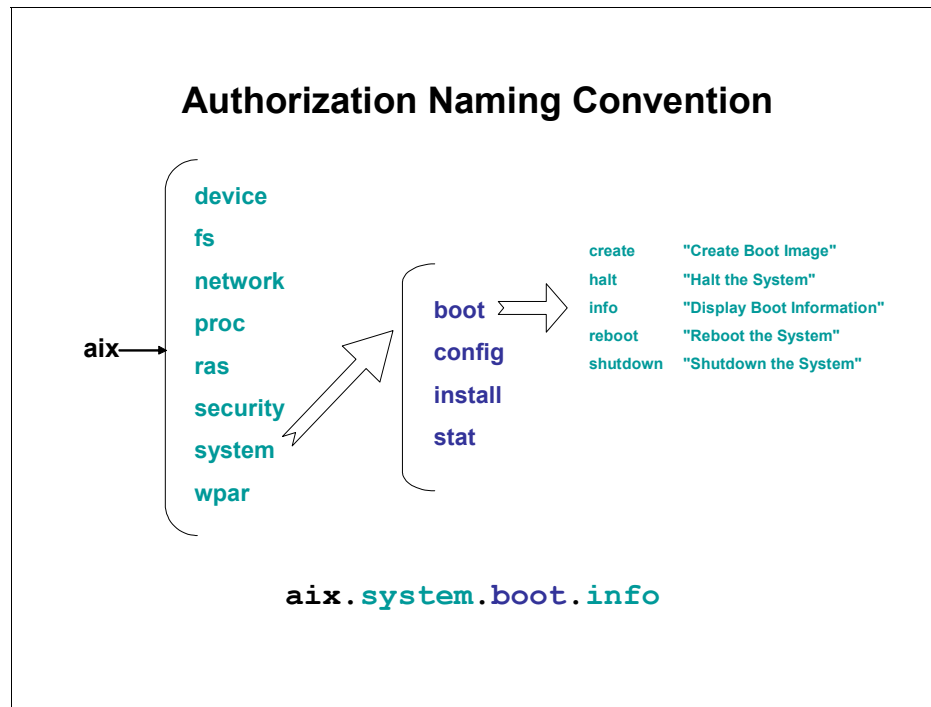
Figure 3-22   Authorization naming convention

Users may define their own hierarchy of authorizations and assign them to roles. The base name of the hierarchy needs to be something other than "aix".

User defined authorizations support the same hierarchy concept as system-defined authorizations.

When creating a user defined authorization, consideration should be given to the following restrictions:

► User defined authorizations must be defined under a new top level parent. It is not possible to create a user defined authorization under the existing "aix." hierarchy. User defined authorizations may not be children of the system defined "aix." hierarchy.

► An authorization name may be up to 63 printable characters with no spaces allowed.

- An authorization may have at most eight parent levels.
- An authorization may have any number of immediate children but can only have one immediate parent. There is no way for two independent authorizations to have the same immediate child.

When creating a user defined authorization, consider the naming convention that will be used. The name of the authorization should include some description or insight into the authorizations purpose.

The following syntax is suggested when creating user-defined authorizations:

vendor_name.product_name.function.function1.function2…..

Table 3-3 further explains the suggested naming format to be used when creating user defined authorizations.

Table 3-3   Suggested syntax for user defined authorization naming

Vendor Name	Identifies the name of the vendor that produces or provides this software module.
Product Name	The high level product name.
Function, Function1, Function2 ..	These strings represent the functions that are being managed using RBAC. Also, when defined, these strings provide a hierarchical representation of how these functions are organized.

An example of this naming convention is ibm.tsm.managment.admin.stgpool. This user defined authorization could potentially be created to represent the administrative management aspects of the IBM Tivoli® Storage Manager storage pool device class.

Using such a format will assist in avoiding conflicts in regards to authorization names across multiple software components. Additionally, this format can provide an insight to the purpose of the authorization.

When creating a user defined authorization, consideration should be given to the following points:

- Is there an existing system defined authorization that includes the appropriate privileged commands? If so, consider whether using the existing authorization would comply with the least privileged principal.
- Will the new authorization belong beneath an existing user-defined authorization hierarchy or is it the first authorization of a new hierarchy?

- If the authorization will require a new hierarchy, what will the structure format be?
- Should an authorization ID be specified when creating the authorization? It is recommended that the `mkauth` command be allowed to generate the authorization ID?

3.7.2 Creating a user defined authorizations

In this section, we will create a user defined authorization named operatorPVI. This authorization will used later by the oper1 user to manage privileged files.

We will also create a user defined role named operator_pvi and assign the operatorPVI authorization to the operator_pvi role.

> **Tip:** In this example, we need only create an authorization for the oper1 user so it will not use a hierarchical authorization tree. If multiple users required different PVI management authorizations, it may be more appropriate to create a hierarchical authorization, such as ibm.pvi.operator.oper1.

AIX Version 6.1 includes several authorization management commands:

`mkauth`   The `mkauth` command creates a new user defined authorization. All parent elements in the specified authorization name must already exist in the authorization database before the new authorization can be created. When the system is operating in enhanced RBAC mode, authorizations created in the authorization database may be assigned to roles immediately, but will not take affect until the Kernel Security Tables have been updated.

`rmauth`   The `rmauth` command removes a user defined authorization. The `rmauth` command will only remove existing user defined authorizations in the authorization database. System-defined authorizations cannot be removed with this command. When the system is operating in enhanced RBAC mode, modifications made to the authorization database are not used for security considerations until the database has been sent to the Kernel Security Tables through the `setkst` command.

`lsauth`   Displays user or system defined authorization attributes.

`chauth`	Modifies attributes for existing user-defined authorizations. System-defined authorizations cannot be modified. When the system is operating in enhanced RBAC mode, modifications made to the authorization database are not used for security considerations until the database has been sent to the Kernel Security Tables through the `setkst` command.

To create the operatorPVI user defined authorization:

1. Log in as your root or your authorization administrative user.

   Figure 3-23 shows the `mkauth` command. In this example, we allow the ID to be system generated and are using the Operator PVI management description in the dfltmsg stanza.

```
root@trinity:/root# mkauth dfltmsg="Operator PVI managment"
operatorPVI
root@trinity:/root# lsauth operatorPVI
operatorPVI id=10018 dfltmsg=Operator PVI managment
root@trinity:/root#
```

Figure 3-23   mkauth command

2. Create the operator_pvi role and assign the operator_pvi authorization.

   Once the operator_pvi role is created, we can display the role, and determine which authorizations are assigned to the role. In this case, we have assigned only the operatorPVI authorization.

   Figure 3-24 shows the `mkrole` and `lsrole` commands.

```
root@trinity:/root# mkrole dfltmsg='Operator PVI management'
authorizations=operatorPVI operator_pvi
root@trinity:/root# lsrole operator_pvi
operator_pvi authorizations=operatorPVI rolelist= groups=
visibility=1 screens=* dfltmsg=Operator PVI management msgcat= id=8
root@trinity:/root#
```

Figure 3-24   The mkrole command

3. Assign the role to the user.

   RBAC roles may be assigned with the `chuser` command.

   The `chuser` command does not append new roles to existing roles. As the oper1 user has an existing role assigned, this existing role will need to be included in the new role value in the `chuser` command.

Figure 3-25 shows the `lsuser` and **chuser** commands.

```
root@trinity:/root# lsuser oper1
oper1 id=208 pgrp=staff groups=staff home=/home/oper1
shell=/usr/bin/ksh gecos=operator login=true su=true rlogin=true
daemon=true admin=false sugroups=ALL admgroups= tpath=nosak ttys=ALL
expires=0 auth1=SYSTEM auth2=NONE umask=22 registry=files
SYSTEM=compat logintimes= loginretries=5 pwdwarntime=0
account_locked=false minage=1 maxage=13 maxexpired=-1 minalpha=1
minother=1 mindiff=1 maxrepeats=2 minlen=6 histexpire=4 histsize=4
pwdchecks= dictionlist= default_roles= fsize=2097151 cpu=-1
data=262144 stack=65536 core=2097151 rss=65536 nofiles=2000
time_last_login=1177326441 tty_last_login=/dev/pts/1
host_last_login=9.41.222.87 unsuccessful_login_count=0
roles=shutdown_reboot
root@trinity:/root#
root@trinity:/root# chuser roles=shutdown_reboot,operator_pvi oper1
root@trinity:/root#
root@trinity:/root# lsuser oper1
oper1 id=208 pgrp=staff groups=staff home=/home/oper1
shell=/usr/bin/ksh gecos=operator login=true su=true rlogin=true
daemon=true admin=false sugroups=ALL admgroups= tpath=nosak ttys=ALL
expires=0 auth1=SYSTEM auth2=NONE umask=22 registry=files
SYSTEM=compat logintimes= loginretries=5 pwdwarntime=0
account_locked=false minage=1 maxage=13 maxexpired=-1 minalpha=1
minother=1 mindiff=1 maxrepeats=2 minlen=6 histexpire=4 histsize=4
pwdchecks= dictionlist= default_roles= fsize=2097151 cpu=-1
data=262144 stack=65536 core=2097151 rss=65536 nofiles=2000
time_last_login=1177326441 tty_last_login=/dev/pts/1
host_last_login=9.41.222.87 unsuccessful_login_count=0
roles=shutdown_reboot,operator_pvi
root@trinity:/root#
```

Figure 3-25   The lsuser and chuser commands

4. Update the KST with the **setkst** command.

   Figure 3-26 shows the **setkst** command.

```
root@trinity:/root# setkst
Successfully updated the Kernel Authorization Table.
Successfully updated the Kernel Role Table.
Successfully updated the Kernel Command Table.
Successfully updated the Kernel Device Table.
root@trinity:/root#
```

Figure 3-26   The setkst command

If the operatorPVI authorization were to be used for privileged commands, the privileged commands could now be assigned to the authorization. As the operatorPVI authorization will be used for privileged file access, no privileged commands will be need to assigned to the authorization.

The privileged file access will be assigned to the operatorPVI authorization in 3.9, "The Privileged File Database" on page 213.

3.8 The Privileged Command Database

This section will introduce Enhanced RBAC privileges and process privilege sets.

3.8.1 Privileges

A *privilege* is a mechanism used to grant a process augmented functionality in system calls. Privileges determine whether a command is eligible to perform an action.

The defining difference between privileges and authorizations is that privileges are associated with specific processes, and authorizations are associated to users through roles.

The concept of privileges is for the most part a kernel-level construct, since the definition and most of the checking occurs there. The assigning of privileges to commands, devices, and processes is performed outside of the kernel, then updated into the kernel with the **setkst** command.

AIX defines privileges as the individual bits of a bit-mask that enforce access control over privileged operations. Over 100 privileges are shipped with AIX V6, providing for a very fine granular control over the privileged operations.

Privileges are assigned to command invocations through the privileged command database and privileges are used to control access to devices through the privileged device database.

When determining access in a system call, the kernel will check that the process has the required privilege bit associated and then grant or deny the request.

Privileges in AIX V6 are predefined and may not be modified, removed, or created.

Privileges, like authorizations, are hierarchical. All privileges begin the "PV_" prefix, which is a textual representation of the privilege bit. Privilege checking is performed by the kernel in a hierarchical manner. When checking for privileges, the system will first check to see if the process has the lowest privilege needed, and then proceed up the hierarchy, checking for the presence of a more powerful privilege.

The PV_ROOT privilege is a special privilege that represents the parent of all privileges except for PV_SU_. A process that is assigned the PV_ROOT privilege will behave as though it has been assigned every privilege on the system except for PV_SU_.

3.8.2 Process Privilege Sets

Process Privilege Sets are used to dynamically restrict or limit privileged operations. Multiple sets of privileges are defined in the kernel to provide for varied controls in regards to privileged operations. Multiple privilege sets allow the operating system to enforce dynamic privilege controls and for applications to manage least privileged principles.

Privileges are associated with a process through the following privilege sets.

Limiting Privilege Set
The Limiting Privilege Set (LPS) defines the maximum or hard limit on privileges for a given process.

The LPS defines the maximum privilege escalation; a process cannot acquire any more privileges than this value using any of the defined interfaces in the system. At any point in time, a process will be restricted to the LPS privileges.

This also means that the rest of the privilege sets will always be subsets of LPS.

Although the LPS is the maximum or hard limit and may not be exceeded, every process will have the right to reduce the LPS. Once a process has reduced the LPS, the new reduced limit becomes the new LPS and cannot be expanded or increased to its original value.

The lowering of the LPS allows a process to restrict the boundaries in regards to associated privileges. For example, a process might reduce the LPS just before executing a user provided custom program.

By default, all the privileges available on the system are set in the LPS for a process.

Maximum Privilege Set

The Maximum Privilege Set (MPS) is the entire set of privileges that the process has been authorized to use. The MPS may include any privilege in the LPS but cannot ever exceed the LPS.

The MPS is not static and has a hard limit only within the limitations of the LPS. For this reason, the value of the MPS could change during the lifetime of a process.

Some examples of when an MPS could change would be:

► When the current process executes another privileged command and then gains related additional privileges.

► If the process has the right privilege, then it can expand the MPS dynamically programmatically.

Effective Privilege Set

The Effective Privilege Set (EPS) is the list of privileges that are currently active for the process. The EPS is always a subset of the process' MPS and is used by the kernel to perform access checks in regards to privileged operations. The EPS is not static and can be manipulated by the process. The EPS can equal the MPS but can never exceed the MPS.

A process may perform dynamic manipulation of the EPS to enforce the least privileged principle.

Inheritable Privilege Set

The Inheritable Privilege Set (EPS) defines privileges that are passed from a parent process to its child process's MPS and EPS.

The IPS may include any privilege in the LPS but cannot ever exceed the LPS.

The IPS can be set in a process in the following ways:

► If the process has the proper privilege, it can expand the IPS programmatically through the setppriv() system call.

► When a privileged command is executed, the privileges specified in the "inheritprivs" attribute associated with the command are assigned to the IPS.

Used Privilege Set

The Used Privilege Set (UPS) describes the privileges that have been used for access checks during the life of the process. The UPS can be used to determine the privileges required by the process.

Whenever the kernel checks if a process has a given privilege, it will store a successful check in the UPS for the privilege.

Workload Partition Privilege Set (WPS)

It is possible to configure a system WPAR to restrict the privileged operations that are allowed in a global environment. The privileged operations allowed in a system WPAR can be controlled through the WPS.

The global environment root may assign a limited set of privileges to a WPAR using WPS. The WPS can be specified in the /etc/wpar/secattrs configuration file or during the start of a WPAR using the `startwpar` command. All processes running in a WPAR will have their LPS equal to their WPS.

Figure 3-27 shows a graphical representation of the Enhanced RABAC Privilege Set.
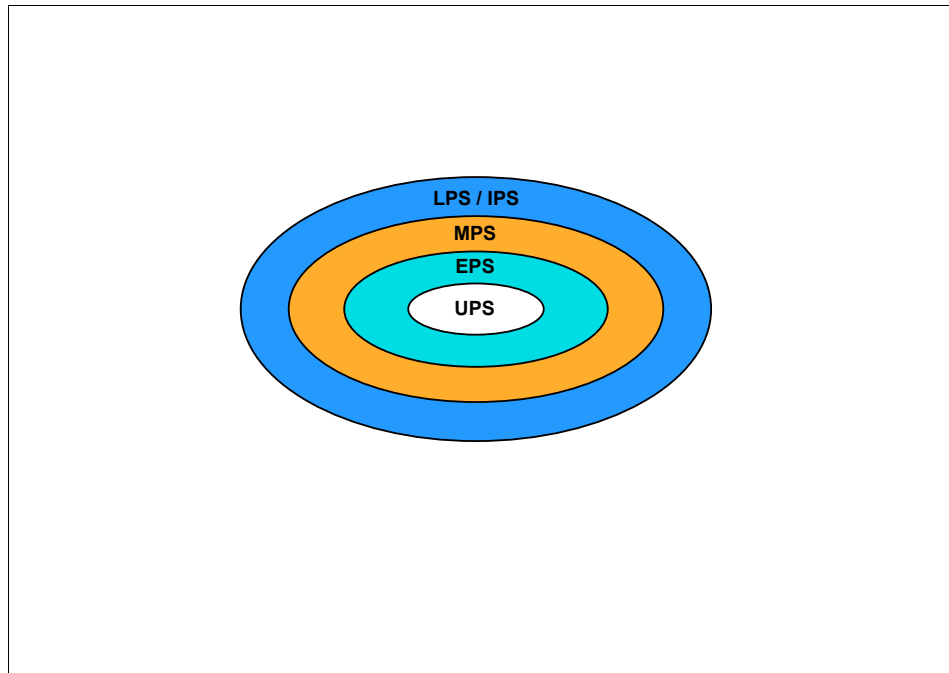
Figure 3-27   Enhanced RBAC privilege sets

Enhanced RBAC provides the following commands to administer privilege sets:

lssectattr   The **lssecattr** command lists the security attributes of one or more commands, devices, or processes. The **lssecattr** can be used to list the LPS, MPS, EPS, IPS, and UPS of an active process.

setsecattr   The **setsecattr** command sets security attributes of the command, device, or process. The **setsecattr** command can be used to modify the LPS, MPS, EPS, and IPS of an active process. The UPS cannot be modified with the **setsecattr** command as the UPS is a read-only attribute.

3.8.3 Privileged commands

As discussed previously, command authorization in AIX has traditionally relied on either DAC or the authorization included directly in the code of the command. Enhanced RBAC utilizes authorizations, roles, and privileges to provide for a more granular security controls method.

The Enhanced RBAC mode provides a framework to enforce authorization checks and grant associated privileges through the Privileged Command Database without requiring changes to the executables on the system or modification of the DAC permissions.

The Privileged Command Database allows the administrator to grant users access and privileges to commands that they would not otherwise be able to execute or would not have the appropriate privilege to perform the task. The Privileged Command Database serves to save the authorization information for a particular command as well as the privileges that are granted to the process if authorization checks succeed.

When the Privileged Command Database is stored locally, it exists in the /etc/security/privcmds file and contains stanzas of information in the form of command versus security attributes.

Some of the key attributes in the Privileged Command Database are:

accessauths   A list of access authorizations that protect the execution of the command. A user with any one of the listed authorizations is allowed to execute the command and perform some or all of the privileged operations contained within.

innateprivs   Innate privileges are privileges assigned to the process if the invoker succeeds the access authorization checks.

authprivs   Authorized privileges are additional privileges assigned to the process if the invoker has the associated authorization. This attribute provides for a more granular control of the command, which can allow a restricted set of users to perform additional privileged operations.

inheritprivs   Inheritable privileges are privileges that the process will pass on to child processes.

secflags   List of security flags.

When a user on an Enhanced RBAC mode system attempts to execute a command, the command is first looked up in the Privileged Command Database.

If the command exists in the database, a check will be performed against the authorizations associated with the user's session and the value of the accessauths attribute for the invoked command. If the session has one of the authorizations listed, then the user will be allowed to execute the command regardless of whether the user passes the DAC execution checks for the command.

Figure 3-28 shows a graphical representation of the Enhanced RBAC command execution process.
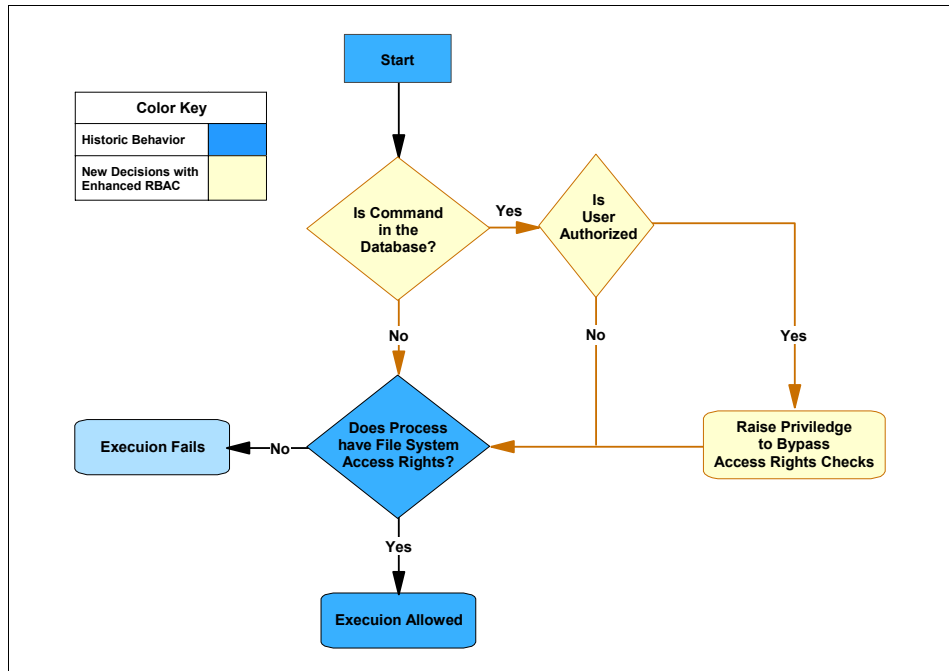
Figure 3-28   Enhanced RBAC command execution

When a command is included in the Privileged Command Database, it is considered to be an RBAC Privileged Command. If a program uses setuid and is not listed in the Privileged Command Database, it may still be classed as privileged, but will not be referred to as an RBAC Privileged command.

If a command does not have an entry in the database, then it is not a RBAC Privileged command and access to it is enforced by DAC and the command itself.

If a command is listed in the privileged command database but the invoker's session does not have an authorization that allows execution of the command, the DAC and UID/GUID checking will still be used to allow execution if those checks succeed.

3.9 The Privileged File Database

In this section, we will introduce the Privileged File Database and the `pvi` command.

3.9.1 Privileged file management with DAC

Traditionally in AIX 5L, many system configuration files are owned by the root user and utilize DAC file permissions settings that do not allow the system configuration files to be modified by general users.

Some system configuration files use a command interface to allow their modification and management. The /etc/inittab file uses DAC to allow only the root user write access to the file. If the file requires an update by a user other than the root user, RBAC could be used to create a role/authorization to allow execution of the inittab management command set, including `chitab`, `rmitab`, `lsitab`, and `mkitab` by the non-root user.

Not all system configuration files in AIX 5L have command interfaces to allow modification of the files. These files are required to have a tool that allows an administrator with the appropriate authorization to directly edit and save a file that they otherwise would not have access to.

3.9.2 Privileged File Management with RBAC

The Privileged File Database contains the list of files that the administrator has defined as requiring privileged authorization. When stored locally, the Privileged File Database is located in the /etc/security/privfiles file. The Privileged File Database maps privileged files to the authorizations required to view or modify them.

The Privileged File Database is used by the `pvi` command to determine the name and authorization modes for any privileged file.

Privileged files may include any system configuration file that does not use its own command interface, as well as additional files that the administrator may wish to restrict access.

The Privileged File Database uses RBAC authorizations as a method of granting or restricting file access.

The files contained in the Privileged File Database should conform to the same `tvi` (trusted `vi`) command and should not be files that are not readable by the `tvi` or `vi` commands.

> **Consideration:** Files must exist before being added to the Privileged File Database. The `pvi` command cannot be used to create a new file.

3.9.3 Privileged File Database restrictions

The Privileged File Database enforces the following restrictions:

- ► All existing restrictions of the `tvi` command are retained in the `pvi` command.
- ► The system must be running in Enhanced RBAC mode.
- ► The file must exist before it can be managed by the Privileged File Database. The `pvi` command cannot be used to create a file.
- ► Only files contained in the Privileged File Database may be edited by the `pvi` command.
- ► The `pvi` command will only allow one file to be opened at a time.
- ► The `pvi` command disables the function for a file to be saved as a different name than the name of the file when opened.
- ► The Privileged File Database cannot be edited with the `pvi` command.
- ► Only regular files can be opened with the `pvi` command. File links cannot be opened with the `pvi` command.
- ► User defined macros are not supported with the `pvi` command.
- ► The Privileged File Database supports only read authorizations (readauth) and write authorizations (writeauth).
- ► When granting writeauth, the readauth is implied, but authorization will not be updated into the readauth stanza in the /etc/security/privfiles file.

3.9.4 Adding a file to the Privileged File Database

The security controls officer has asked us to allow the oper1 user to be given read/write access to the /etc/hosts file. We will create a user defined role and authorization and then add the /etc/hosts file to the Privileged File Database, allowing the oper1 user to read and write to the /etc/hosts file.

The /etc/hosts file has the DAC permissions allowing the oper1 user, and all other users other then root or system group members, read permission only.

Figure 3-29 shows the existing DAC permissions of the /etc/hosts file.

```
oper1@trinity:/home/oper1# ls -ltra /etc/hosts
-rw-rw-r--   1 root     system          1853 May 04 01:40 /etc/hosts
oper1@trinity:/home/oper1#
```

Figure 3-29   DAC permissions of the /etc/hosts file

To add a file to the Privileged File Database:

1. Create or determine an authorization and role that will be used for Privilege File access.

   The Privileged File Database uses RBAC authorizations to authenticate the read or write access controls to the file.

   For a file to be added to the Privileged File Database, the administrator must know:

   – The File name.

   – The RBAC authorization for the read authorization (readauth). readauth is not mandatory if writeauth is specified. readauth may be specified as a list of authorizations.

   – The RBAC authorization for the write authorization (writeauth), if required. writeauth may be specified as a list of authorizations.

   In this case, we will use the operatorPVI authorization and operator_pvi role that we created earlier in 3.6, "User defined roles" on page 191.

2. Using the `setsecattr` command, we add the /etc/hosts file to the Privileged File Database.

   We will define the operatorPVI authorization write access to the /etc/hosts file.

In Figure 3-30, we use the **setsecattr** command to add the /etc/hosts file into the Privileged File Database and allow the authorization operatorPVI write access to the file. Write access (writeauth) will also allow read access (readauth) to the /etc/hosts file.

```
root@trinity:/root# setsecattr -f writeauths=operatorPVI /etc/hosts
root@trinity:/root#
```

Figure 3-30   The setesattr command

3. Update the Kernel Security Tables with the **setkst** command.

If the oper1 user were to attempt to use the **pvi** command without the KST being updated, the **pvi** command would fail as the security checks in the KST would not find a valid entry for readauth or writeauth for the oper1 user.

Figure 3-31 shows the oper1 user executing the **pvi** command prior to updating the KST.

```
$ pvi /etc/hosts
"/etc/hosts"pvi: 0602-193 Authorization check failure.
$
```

Figure 3-31   The pvi command accesses the /etc/hosts file with KST updated

4. The oper1 user may now read and write to the /etc/hosts file with the **pvi** command.

If the oper1 user were to try and modify and save the /etc/hosts file with the **vi** command, the /etc/hosts DAC file permissions would be used to determine whether the oper1 user has the authority to save changes to the file.

The oper1 user does not have DAC write access to the /etc/hosts file, so the attempt to save the /etc/hosts file would not be successful.

Figure 3-32 shows the oper1 user editing the /etc/hosts file with the **vi** command. The file can be viewed, but an attempt to save the data fails, as the DAC file permissions allow only read access for user oper1.

```
# @(#)47        1.1  src/bos/usr/sbin/netstart/hosts, cmdnet, bos540 7/24/91 10:
00:46
# IBM_PROLOG_BEGIN_TAG
# This is an automatically generated prolog.
#
# bos540 src/bos/usr/sbin/netstart/hosts 1.1
#
# Licensed Materials - Property of IBM
#
# (C) COPYRIGHT International Business Machines Corp. 1985,1989
# All Rights Reserved
#
# US Government Users Restricted Rights - Use, duplication or
# disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
#
# IBM_PROLOG_END_TAG
#
# COMPONENT_NAME: TCPIP hosts
#
# FUNCTIONS: loopback
#
# ORIGINS: 26  27
#
"/etc/hosts"ex: 0602-065 The file has read permission only.
```

Figure 3-32   The /etc/hosts file edited by oper1 with the vi command

When the /etc/hosts file is edited with the **pvi** command, the outcome is different.

When using the **pvi** command, the user oper1 may modify and save the /etc/hosts file.

Figure 3-33 shows the oper1 user editing the /etc/hosts file with the **pvi** command. The file can be viewed, modified, and saved. Even though the DAC file permissions allow only read access for user oper1.

```
00:46
# IBM_PROLOG_BEGIN_TAG
# This is an automatically generated prolog.
#
# bos540 src/bos/usr/sbin/netstart/hosts 1.1
#
# Licensed Materials - Property of IBM
#
# (C) COPYRIGHT International Business Machines Corp. 1985,1989
# All Rights Reserved
#
# US Government Users Restricted Rights - Use, duplication or
# disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
#
# IBM_PROLOG_END_TAG
#
# COMPONENT_NAME: TCPIP hosts
#
# FUNCTIONS: loopback
#
# ORIGINS: 26  27
#
"/etc/hosts" 54 lines, 1853 characters
oper1@trinity:/home/oper1#
```

Figure 3-33   The /etc/hosts file edited by oper1 with the pvi command

3.10 The Privileged Device Database

This section discusses the Privileged Device Database.

3.10.1 Privileged device management with RBAC

The Privileged Device Database is similar in concept to the Privileged File Database, but instead of controlling access to a file, the Privileged Device Database is used to control access to a device.

The Privileged Device Database can be used to provide further control access to a database that could be managed through traditional device access controls. When the database is stored locally, it is contained in the /etc/security/privdevs file.

The Privileged Device Database stores the privilege information for a device for read or write operations in the following attributes:

readprivs   Lists privileges that are allowed to read from the device. When a read request is opened for a privileged device, the open is allowed only if one of the privileges specified in readprivs exists in the Effective Privilege Set (EPS) for that process.

writeprivs   Lists privileges that are allowed to write to the device. When a write request is opened for a privileged device, the open is allowed only if one of the privileges specified in writeprivs exists in the Effective Privilege Set (EPS) for that process.

Including a device in the Privileged Device Database may be performed with the `lssecattr` and `setsecattr` commands.

Before including a device in the Privileged Device database, a thorough investigation of the commands and applications that need to access the device must be performed to ensure that the proper privileges are specified.

3.11 Securing the root user

The following section will discuss disabling the root user when running in Enhanced RBAC mode.

3.11.1 Choosing to secure the root user

When the AIX system is operating in Enhanced RBAC mode, it is possible to configure the system such that the root user has no super user privileges and is disabled so that no login access will be available to the root user account.

Traditionally in AIX, the root user's UID value of 0 has been treated as a privileged UID by the operating system and allowed the root user to bypass the enforced security checks. By disabling the root user, these operating system checks are effectively removed.

With the root user disabled, the root user would be restricted from accessing the system, although it would still retain DAC ownership of files if the account were able to be accessed. Though the root user can still own files objects, the root user cannot be accessed through the **su** command or logged into remotely or from the defined system console.

As traditional UNIX administration relies on the root user being enabled for privileged commands; an attacker will expect the root user to be enabled and may attempt to concentrate their attack efforts on the root user.

An attacker may attempt to gain access to the root user, knowing that if the root user's integrity is compromised, the attacker will be free to execute any privileged command with malicious intent. If unauthorized root user access is obtained, the attacker may cause extensive and unrestricted damage to the system. This intentional damage is known as *malicious root*.

In an Enhanced RBAC system, where the root user is disabled, the damage an attacker may cause can be minimized, since the root user is disabled. Were the attacker to compromise whatever network security is in place and receive a login prompt, the attacker cannot access the root account either remotely or at the console or through the `su` command.

After disabling the root user, system administration must be performed by users other than the root user. Access to privileged command and files that are owned by the root user will need to be performed by one or more other user accounts. One such way would be to utilize the isso, so, and sa roles that are predefined with AIX V6.

> **Important:** The isso, so, and sa roles may not include all privileged commands or authorizations required to manage the system.
>
> A careful analysis of the system and the applications being used on the system should be performed before attempting to disable the powers of the root user.

3.11.2 Disabling the root user

The powers of the root user can be disabled through the `setsecconf` command.

In this scenario, we have first assigned the isso role to the oper1 user, so that we may test the oper1 user execution of the privileged commands required to disable or enable the root user mode.

The oper1 user still has the shutdown_reboot role, so the oper1 user will be able to execute the `reboot` and `shutdown` commands to reboot the system.

Execute the following commands and then reboot the system to disable the powers of the root user:

1. Log in as the oper1 user and execute the `swrole` command to the isso role:

   ```
   swrole isso
   ```

2. After authenticating, set the runmode to configuration mode:

   ```
   setrunmode -c
   ```

3. Execute the **setseconf** command to disable the root user:

   ```
   setsecconf -o root=disable
   ```

4. Reboot the system.

   In this case, we need to first swap roles to the shutdown_reboot role and authenticate before executing the **shutdown -Fr** command:

   ```
   swrole shutdown_reboot
   shutdown -Fr
   ```

The system will now reboot and will restart with the root=disable mode.

Figure 3-34 shows disabling the root user in Enhanced RBAC mode.

```
oper1@trinity:/home/oper1# swrole isso
oper1's Password:
oper1@trinity:/home/oper1#
oper1@trinity:/home/oper1# setrunmode -c
System is already running in CONFIGURATION MODE.
oper1@trinity:/home/oper1# setsecconf -o root=disable
OPERATIONAL MODE Security Flags
ROOT                      :   DISABLED
Changes take effect at next boot time.
oper1@trinity:/home/oper1# swrole shutdown_reboot
oper1's Password:
oper1@trinity:/home/oper1# shutdown -Fr

SHUTDOWN PROGRAM
Thu May  3 06:49:51 MST 2007
0513-044 The sshd Subsystem was requested to stop.

Wait for 'Rebooting...' before stopping.
```

Figure 3-34   Disabling the root user in Enhanced RBAC mode

Figure 3-35 shows the console output when the server is rebooting with the root user disabled.

```
System initialization completed.
TE=OFF
CHKEXEC=OFF
CHKSHLIB=OFF
CHKSCRIPT=OFF
CHKKERNEXT=OFF
STOP_UNTRUSTD=OFF
STOP_ON_CHKFAIL=OFF
TSD_FILES_LOCK=OFF
TSD_LOCK=OFF
TEP=OFF
TLP=OFF
Successfully updated the Kernel Authorization Table.
Successfully updated the Kernel Role Table.
Successfully updated the Kernel Command Table.
Successfully updated the Kernel Device Table.
OPERATIONAL MODE Security Flags
ROOT                      :   DISABLED
System runtime mode is now OPERATIONAL MODE.
Setting tunable parameters...complete
Starting Multi-user Initialization
```

Figure 3-35   Console reboot with root user disabled

After executing the **setsecconf** command and disabling the root user, the root user is still a valid user identity on the system but cannot be accessed with the **su** command or through a login from remote or local consoles.

3.11.3 Considerations when disabling the root user

When choosing to disable the root user, any root owned processes are no longer assigned any special powers or privileges.

In an Enhanced RBAC system with the root user disabled, any command that needs to perform privileged operations should be added to the Privileged Command Database and assigned the appropriate privileges. This would include any setuid applications or commands that are owned by the root user; as with the root user disabled, the checking mechanism for the UID 0 will cease to bypass enforced security checks.

Executing setuid applications or commands in a root disabled environment would likely cause the command or application to perform with an unreliable or unsuccessful completion.

When considering disabling the root user, ensure that any operational or administrations tasks that would performed by the "root" user have been identified and these tasks have been assigned to one or more user accounts.

To disable or enable the root user requires a system reboot, so if it was discovered that a command or file was unable to be accessed or executed once the root user was disabled, the enablement of the root user would disrupt the operation of the server.

In this scenario, the root user may be enabled by switching roles to the isso role and executing the `setsecconf -o root=enable` command and rebooting the system.

Figure 3-36 shows enabling the root user in Enhanced RBAC mode.

```
oper1@trinity:/home/oper1# swrole isso
oper1's Password:
oper1@trinity:/home/oper1#setrunmode -c
System runtime mode is now CONFIGURATION MODE.
oper1@trinity:/home/oper1# setsecconf -o root=enable
OPERATIONAL MODE Security Flags
ROOT                      :    ENABLED
Changes take effect at next boot time.
oper1@trinity:/home/oper1# swrole shutdown_reboot
oper1's Password:
oper1@trinity:/home/oper1# shutdown -Fr

SHUTDOWN PROGRAM
Thu May  3 08:57:25 MST 2007
0513-044 The sshd Subsystem was requested to stop.

Wait for 'Rebooting...' before stopping.
```

Figure 3-36   Enabling the root user in Enhanced RBAC mode

Important: A careful analysis of the system and the applications being used on the system should be performed before attempting to disable the powers of the root user.

3.11.4 Summary of root disable mode with Enhanced RBAC

To summarize the disabling of the root user in Enhanced RBAC:

► The root user should only be disabled if operating in Enhanced RBAC mode.

► Disabling the root user is not a mandatory requirement for operating in Enhanced RBAC mode.

► setuid applications or commands would likely cease to function in a root disabled environment. These commands or applications would likely need addition to the Privileged Command Database, and then further tested to ensure their operation.

► A careful analysis of the system and the applications being used on the system should be performed before attempting to disable the powers of the root user. Thorough testing should be carried out before putting the system into production.

► The isso, sa and so roles may not include all privileged commands or authorizations required to manage the system.

► Any privileged command that may be required to administer the system should be added to the privileged command database.

► Any privileged file owned by the root user that may require access by a non-privileged user may need to be added to the Privileged File Database.

Note: The root disable mode is specific to each WPAR. This means that the root user functionality can be selectively disabled/enabled on a given WPAR. By default, when WPARs are created, root will be enabled regardless of whether root is disabled in the global environment or not. Modification to this flag requires a reboot of the corresponding WPAR.

3.11.5 Using the FPM command to reduce SetUID programs

The File Permission Manager (FPM) command may be used to manage the setuid and setgid DAC permissions on executable commands.

By combining the FPM with Enhanced RBAC, an administrator may choose to modify the setuid or setgid bit on one or more executables and instead execute these programs as RBAC privileged commands.

By using the `fpm` command to modify the executables to remove the s-bit, a non-privileged user would no longer be able to assume the effective UID of 0 and execute commands as the root user. The user would instead execute an RBAC privileged command using an Enhanced RBAC authorization and role.

Enhanced RBAC does not require that the administrator use the FPM to remove s-bits on setuid and setgid commands, though using the FPM in this manner may be an attractive option to administrators whose environment demands that s-bit execution be kept to a minimum.

As with any modification of the operating system environment, careful analysis of the system and the applications being used on the system should be performed before attempting to modify the s-bit executables.

3.12 Enhanced RBAC and WPAR

This section will discuss the use of Enhanced RBAC and the Workload Partition (WPAR).

Restrictions
The following restrictions apply using RBAC with WPAR:

► Only Enhanced RBAC mode is supported when using WPAR.

► Enhanced RBAC is supported only on System WPAR.

► The system defined authorizations are contained in the Global Environment. Each System WPAR KST will utilize the Global Environment system defined authorizations.

► Each WPAR has its own WPAR KST, which resides in the Global Environment kernel.

► Any WPAR that has a privilege limited by the use of the WPS will not be authorized to access that privilege, even if present in the WPAR KST.

Figure 3-37 shows the usage of Enhanced RBAC and System WPAR authorizations, utilizing the Global Environment system defined authorizations.
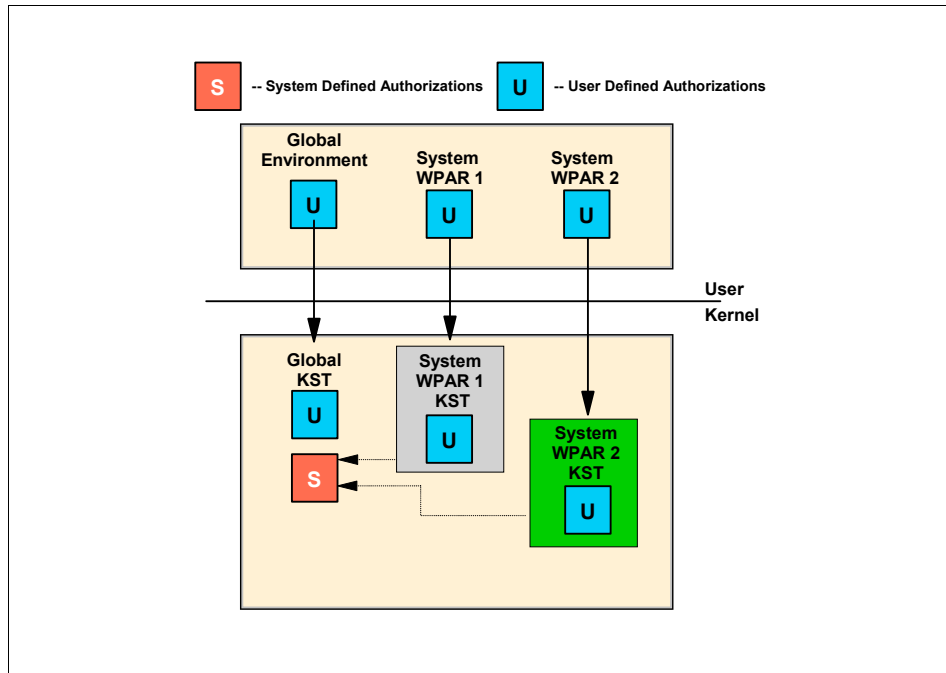
Figure 3-37 System WPAR and Enhanced RBAC

3.13 Migrating to Enhanced RBAC

This section discuss migration to Enhanced RBAC mode from a pre-existing RBAC environment.

3.13.1 Migrating authorizations

The Legacy RBAC product release in versions prior to AIX V6 consisted of a limited predefined set of authorizations. These Legacy RBAC authorizations were not defined in a file on the system, but could be readily assigned to roles.

In AIX V6, Enhanced RBAC supports these Legacy RBAC authorizations within the new Enhanced RBAC framework as user-defined authorizations. These user-defined authorizations are provided by default in the RBAC authorization security database located in /etc/security/authorizations.

Since AIX V6 uses a new naming convention for authorizations, any checks for the old authorization names in AIX have been modified to additionally check for the new Enhanced RBAC authorizations.

Table 3-4 lists the Legacy Mode predefined authorizations and the corresponding Enhanced Mode system-defined authorizations.

Table 3-4   Authorizations in Legacy Mode and Enhanced Mode RBAC

Existing Legacy Mode authorization	Enhanced Mode authorization
Backup	aix.fs.manage.backup
Diagnostics	aix.system.config.diag
DiskQuotaAdmin	aix.fs.manage.quota
GroupAdmin	aix.security.group
ListAuditClasses	aix.security.audit.list
PasswdAdmin	aix.security.passwd
PasswdManage	aix.security.passwd.normal
UserAdmin	aix.security.user
UserAudit	aix.security.user.change
RoleAdmin	aix.security.role
Restore	aix.fs.manage.restore

Important: After a migration, the system administrator must verify that the authorizations and roles are defined as desired for the environment.

3.13.2 Role migration

When an updating an existing system operating on a version of AIX prior to AIX V6 through a migration install, migration of the /etc/security/roles file will attempt to update the file for the new Enhanced RBAC functionality while maintaining the current role abilities.

Role definitions in the file will be preserved and modified to include a unique role ID so that the role may function in the new Enhanced RBAC framework.

Any authorizations that appear in the /etc/security/roles file and are not known predefined authorizations will be considered user-defined authorizations. During migration, these authorization names will be added as entries into the local /etc/security/authorizations authorization database. In addition to migration of the Legacy RBAC role definitions, the new predefined roles will be appended to the file.

> **Important:** After a migration, the system administrator must verify that the authorizations and roles are defined as desired for the environment.

3.14 RBAC remote database support

This section will discuss remote database support for Enhanced RBAC using LDAP.

3.14.1 Prerequisites to using LDAP as an RBAC database repository

Prior to using LDAP as an RBAC database repository, the following prerequisites must be completed:

- ► An LDAP server must be configured and available to host the RBAC security database(s).
- ► There must be network connectivity between the LDAP server and LDAP client.
- ► The LDAP client must have the LDAP client fileset installed.
- ► The LDAP client must be operating in Enhanced RBAC mode.

To host the RBAC security database on an LDAP server, the RBAC database repository needs to be populated into the LDAP server.

AIX V6 provides the `rbactoldif` command, which can be used to read the data in the local RBAC databases and output them in a format suitable for LDAP. The output generated by `rbactoldif` can be saved to a file and then used to populate the LDAP server.

Once the RBAC security database data has generated with the `rbactoldif` command, the data can be uploaded to the LDAP server with the `ldapadd` command.

The following databases on the local system will be used by the **rbactoldif** command to generate the RBAC security database data for LDAP:

- ▶ /etc/security/authorizations
- ▶ /etc/security/privcmds
- ▶ /etc/security/privdevs
- ▶ /etc/security/privfiles
- ▶ /etc/security/roles

Consideration should be given as to where to store the RBAC security database data in LDAP. We suggest that the RBAC data in LDAP be placed under the same parent DN as the user and group data. The ACLs on the data should then be adjusted as appropriate for the security policy that has been chosen.

> **Restriction:** Authorization databases stored in LDAP will only contain the user-defined authorizations. System-defined authorizations cannot be stored in LDAP and will remain local to each client system.

3.14.2 LDAP client configuration for RBAC

An AIX V6 system must be configured as an LDAP client in order to make use of RBAC security database data stored in LDAP. The **mksecldap** command can be used to configure the system as an LDAP client.

The **mksecldap** command will dynamically search the specified LDAP server to determine the location of the authorization, role, and privileged command/device/file data, and save the results to the /etc/security/ldap/ldap.cfg file.

After successfully configuring the system as an LDAP client through the **mksecldap** command, the system must be further configured to enable LDAP as a lookup domain for RBAC data. The /etc/nscontrol.conf file must be modified to include LDAP in the secorder attribute for databases that are to be stored on LDAP.

Once the client system has been configured both as an LDAP client and as a lookup domain for RBAC data, the client daemon **secldapclntd** can be configured to periodically retrieve the RBAC data from LDAP and send the data to the Kernel Security Tables (KST). The **secldapclntd** updates the KST by executing the **setkst** command.

The time period used by the **secldapclntd** daemon to retrieve the RBAC data from LDAP is configurable through the rbacinterval attribute in /etc/security/ldap/ldap.cfg. If the /etc/nscontrol.conf file is using the LDAP and local file database, the **setkst** command will retrieve a merged copy of the entries for a given database as defined in the /etc/nscontrol.conf file and then load the resulting data into the client Kernel Security Tables.

By default, the value of the rbacinterval attribute is 3600, which denotes that the automatic update of the KST is set to run each hour. If the rbacinterval attribute is changed, then the **restart-secldapclntd** command should be run to restart the **secldapclntd** daemon.

> **Note:** If the rbacinterval is 0, then the KST is only updated when an administrator manually executes the **setkst** command on the client system.

3.14.3 Name service control file

A system administrator may define the RBAC data to reside strictly in local files, strictly in LDAP, or for a merge of the data in local files and LDAP. This option may be set up to occur by configuring the lookup domain for each RBAC security database in the name service control file /etc/nscontrol.conf.

The /etc/nscontrol.conf file individually specifies the search order for the authorization, role, privileged command, device, and file databases. It is possible to have a mixture of local file, LDAP, or combined domains for each of the five LDAP security database.

The search order for a database is specified in the /etc/nscontrol.conf file through the secorder attribute, which is a comma separated list of domains. The domain would be either LDAP or file, with file referring to the local database files in the /etc/security directory on the LDAP client.

Figure 3-38 shows an example of the configuration for the authorization database in the /etc/nscontrol.conf file.

```
authorizations:
secorder = LDAP,files
```

Figure 3-38   Example of the /etc/nscontrol.conf file

The example in Figure 3-38 on page 230 specifies that any domain lookup queries on the authorizations security database should first search in LDAP. If an authorization entry was not found in the LDAP database, then the local database would be searched.

The collection of authorizations available to the client system will be the merge of the authorizations provided by LDAP and those provided in the local files. The merge is not a simple combination of the values from the two domains but rather a union of the values. The merge comprises of the RBAC security database from the first lookup domain, then any additional RBAC security database from the remaining lookup domain.

For the configuration in Figure 3-38 on page 230, all authorizations from LDAP would be included and then only unique authorizations from the /etc/security/authorizations local file would be added to the result.

RBAC database modifications and deletions are attempted on the first domain listed and only attempted on subsequent domain if the entity is not found in the first domain.

In Figure 3-38 on page 230, the modification or deletion would first be attempted on the LDAP database. If the LDAP database did not contain the authorization, only then would the modification or deletion occur on the local files database.

New RBAC database entries are always created in the first domain listed in the secorder attribute. In Figure 3-38 on page 230, the creation of a new authorization will occur in the LDAP database.

In case there is no entry for a database in the /etc/nscontrol.conf file or the file does not exist, queries and modifications on the RBAC database will only be performed in the local files database.

It is possible to use the LDAP server for only selected security databases. The configuration for an individual database may be set through the `chsec` command. The configuration for a database may be listed through the `lssec` command.

To configure authorization data to be retrieved from LDAP first then local files, execute the following command:

```
chsec -f /etc/nscontrol.conf -s authorizations -a secorder=LDAP,files
```

The configuration in the /etc/nscontrol.conf file controls both library and command-line interfaces. Applications may retrieve the current value of the secorder attribute for a database through the getsecorder() interface. The value of the secorder attribute may be overridden for the process by using the setsecorder() interface.

3.14.4 RBAC Command Enablement for LDAP

All of the RBAC database management commands are enabled to follow the database lookup order configured in the /etc/nscontrol.conf file.

The database management commands will, by default follow the lookup as defined in the secorder attribute in the /etc/nscontrol.conf file.

If required, the lookup order may be specified by using the -R option on any command, and then specifying the database (LDAP or files).

Specifying the -R option for an RBAC management command forces the operation to occur on the specified domain and overrides the configuration in /etc/nscontrol.conf.

The following RBAC database management commands are enabled for remote domain support:

► `mkauth`, `chauth`, `lsauth`, and `rmauth`

► `mkrole`, `chrole`, `lsrole`, and `rmrole`

► `setsecattr`, `lssecattr`, and `rmsecattr`

► `setkst`

Figure 3-39 shows the `lsrole` database management command used with the -R option to list the local files database instead of the LDAP database.

```
root@trinity:/root# lsrole -R files operator_pvi
operator_pvi authorizations=operatorPVI rolelist= groups=
visibility=1 screens=* dfltmsg=Operator PVI management msgcat= id=8
root@trinity:/root#
```

Figure 3-39   Using the -R flag with the lsrole database management command

The `setkst` command is enabled to honor the configuration contained within /etc/nscontrol.conf. The `setkst` command will retrieve a merged copy of the entries for a given database as defined in the /etc/nscontrol.conf file and then load the resulting data into the client Kernel Security Tables.

3.15 RBAC scenarios

In this section, we will discuss two RBAC scenarios.

Scenario one deals with the division of RBAC roles and defining a privileged command.

Scenario two deals with hosting the Enhanced RBAC security database on an LDAP server.

3.15.1 Scenario 1: Division of roles

In this scenario, an administrator has been asked to define an Enhanced RBAC role that can be used by a storage support specialist to maintain the storage environment on an AIX V6 system.

The storage support specialist is responsible for creating file systems and managing the physical volumes that are assigned to the system.

The storage support specialist has listed the privileged commands that are required for day to day and service support. These commands are:

► `rmdev`
► `mkdev`
► `cfgmgr`
► `mklv`
► `rmlv`
► `crfs`
► `mount`
► `umount`

By configuring a role and authorization for only the commands that the storage support specialist requires to perform these commands, the administrator is able to:

► Leave the root password unchanged.

► Provide the storage support specialist the authorization to use only a limited number of privileged commands. This will limit the access to sensitive data available to a non-privileged user.

► Minimize the potential for misuse of the root user.

► Minimize the potential for system definitions to be modified, whether by accident or malicious intent.

► Comply with the least privileged principle.

To create the storage support specialist role, we must first:

1. Use the `lssecattr` command to determine if the commands requested by the storage support specialist currently exist in the privileged command database. If they do exist, take note of the value in the accessauth stanza, so that this authorization can be added to a role.

   Figure 3-40 shows the `lssecattr` command used with the accessauths flag to display the authorization for each privileged command.

   The `lssecattr` command output produces an error that states that the `cfgmgr` command is not defined in the privileged command database, so in step 2 of this scenario, the `cfgmgr` command will be defined into the privileged command database.

```
root@trinity:/root# lssecattr -c -a accessauths /usr/sbin/rmdev
/usr/sbin/rmdev accessauths=aix.device.manage.remove
root@trinity:/root# lssecattr -c -a accessauths /usr/sbin/mkdev
/usr/sbin/mkdev accessauths=aix.device.manage.create
root@trinity:/root# lssecattr -c -a accessauths /usr/sbin/mklv
/usr/sbin/mklv accessauths=aix.lvm.manage.create
root@trinity:/root# lssecattr -c -a accessauths /usr/sbin/rmlv
/usr/sbin/rmlv accessauths=aix.lvm.manage.remove
root@trinity:/root# lssecattr -c -a accessauths /usr/sbin/crfs
/usr/sbin/crfs accessauths=aix.fs.manage.create
root@trinity:/root# lssecattr -c -a accessauths /usr/sbin/mount
/usr/sbin/mount accessauths=aix.fs.manage.mount
root@trinity:/root# lssecattr -c -a accessauths /usr/sbin/umount
/usr/sbin/umount accessauths=aix.fs.manage.unmount
root@trinity:/root# lssecattr -c -a accessauths /usr/sbin/cfgmgr
1420-012 "/usr/sbin/cfgmgr" does not exist in the privileged command
database.
```

Figure 3-40   The lssecattr command using the accessauth flag

The authorizations displayed in the accessauths field will be used later in step 5 on page 239 when the user defined role is created.

2. In step 1, we saw that the `cfgmgr` command is not defined in the privileged command database. If a command does not exist in the privileged command database, then it must be defined in the RBAC privileged command database before the command can be used as an RBAC privileged command.

To add a command to the privileged command database:

a. Create a user-defined authorization.

In this scenario, we will create the ibm.aix.management.cfgmgr authorization.

Figure 3-41 shows the `mkauth` command being used to create the user-defined authorization ibm.aix.management.cfgmgr.

```
root@trinity:/root# mkauth dfltmsg='IBM Vendor Software Parent
Authorization' ibm
root@trinity:/root# mkauth dfltmsg='IBM AIX operating system
Authorization' ibm.aix
root@trinity:/root# mkauth dfltmsg='IBM AIX operating system
management Authorization' ibm.aix.management
root@trinity:/root# mkauth dfltmsg='IBM AIX operating system cfgmgr
Authorization' ibm.aix.management.cfgmgr
```

Figure 3-41   The mkauth command

> **Note:** As this is a new authorization and there is no IBM AIX
> user-defined authorization, we first create the parent levels in the
> hierarchy.

b. Once the user-defined authorization is defined, use the `tracepriv` command to determine the privileges required for the command not in the privileged command database and add them to the privileged command database.

To define the `cfgmgr` command in the privileged command database, we first need to determine which privileges are required for the command to execute successfully. To determine these privileges, we use the `tracepriv` command.

The `tracepriv` command may be run with the privilege listing output being output to the console or redirected to a text file. In this scenario, we will redirect the output to the /tmp/cfgmgr.tracepriv.output file.

Figure 3-42 shows the **tracepriv** being executed against the **cfgmgr** command. The -f flag is used to follow **forks**. The -e flag is used to follow **execs**.

```
root@trinity:/root# tracepriv -ef -o /tmp/cfgmgr.tracepriv.output
cfgmgr
root@trinity:/root#
```

Figure 3-42   The tracepriv command, redirected to /tmp/cfgmgr.tracepriv.output cfgmgr

Note: The **tracepriv** command may also be run by a non-root user that has had its shell modified with EPS and MPS set to PV_ROOT. This method can be used to ensure that GETUID() checks are properly traced.

Once the **cfgmgr** command has completed, the privileges required by the **cfgmgr** command will be listed in the /tmp/cfgmgr.tracepriv.output file.

Figure 3-43 shows the first page of output from the **tracepriv** command, written to the /tmp/cfgmgr.tracepriv.output file.

```
root@trinity:/root# more /tmp/cfgmgr.tracepriv.output

352496: Used privileges for /etc/methods/cfgprobe:
  PV_DEV_QUERY                        PV_SU_UID

352498: Used privileges for /etc/methods/defsys:
  PV_DAC_O

340196: Used privileges for /usr/lib/boot/bin/chcod_chrp:
  PV_SU_UID

340198: Used privileges for /usr/sbin/lsitab:
  PV_SU_UID
```

Figure 3-43   The tracepriv output from the cgfmgr command

Note: The entire list of privileges required for the **cfgmgr** command to execute have not been listed in Figure 3-43 due to the length of the output. The commands will be listed later in the scenario when discussing the **setsecattr** command.

By reading the **tracepriv** output, we can determine the names of the 15 privileges that the **cfgmgr** command uses to execute. These privileges will be used in the **setsecattr** command to define the **cfgmgr** command in the privileged command database.

c. Add the command to the privileged command database with the **setsecattr** command.

The privileges that were listed by the **tracepriv** command will now be assigned to the **cfgmgr** command as innate privileges in the innateprivs stanza.

The authorization ibm.aix.management.cfgmgr that was defined in step 1 on page 234 will now be assigned to the **cfgmgr** command in the authorization stanza.

Figure 3-44 shows the **setsecattr** command being used to define the **cfgmgr** command in the privileged command database.

```
root@trinity:/root# setsecattr -c
innateprivs=PV_AU_ADD,PV_AU_PROC,PV_DAC_O,PV_FS_CHOWN,PV_SU_UID,PV_T
CB,PV_DAC_R,PV_DEV_LOAD,PV_DEV_QUERY,PV_DEV_CONFIG,PV_KER_ACCT,PV_NE
T_PORT,PV_PROC_PRIV,PV_NET_CNTL,PV_PROC_SIG
accessauths=ibm.aix.management.cfgmgr /usr/sbin/cfgmgr
root@trinity:/root#
```

Figure 3-44   Defining the cfgmgr command with setsecattr

Once the **cfgmgr** command is defined in the privileged command database, we can display the status with the **lssecattr** command.

Figure 3-45 shows the **lssecattr** command.

```
root@trinity:/root# lssecattr -F -c /usr/sbin/cfgmgr
/usr/sbin/cfgmgr:
accessauths=ibm.aix.management.cfgmgr
innateprivs=PV_AU_ADD,PV_AU_PROC,PV_DAC_O,PV_FS_CHOWN,PV_SU_UID,PV_T
CB,PV_DAC_R,PV_DEV_LOAD,PV_DEV_QUERY,PV_DEV_CONFIG,PV_KER_ACCT,PV_NE
T_PORT,PV_PROC_PRIV,PV_NET_CNTL,PV_PROC_SIG

root@trinity:/root#
```

Figure 3-45   Displaying the privileged command cfgmgr with lssecattr

The **cfgmgr** command has now been added to the privileged command database and is now considered an RBAC privileged command.

3. Determine if there is an existing role that may be used that contains the authorizations required to perform the storage support specialist's command list. Be sure you comply with the least privileged principle.

   If there is no existing role on the system that contains the authorizations required, then create a user defined role.

   In this scenario we will define the storage_support role and assign to it the authorization ibm.aix.management.cfgmgr as well as the authorizations listed in Figure 3-40 on page 234.

   Figure 3-46 shows the usage of the **mkrole** command when defining the `config_manager` role.

```
root@trinity:/root# mkrole
authorizations=ibm.aix.management.cfgmgr,aix.device.manage.remove,ai
x.device.manage.create,aix.fs.manage.create,aix.fs.manage.remove,aix
.fs.manage.create,aix.fs.manage.mount,aix.fs.manage.unmount
storage_support
root@trinity:/root#
```

Figure 3-46   The mkrole command

The attributes of the user-defined role can be displayed with the **lsrole** command

Figure 3-47 shows the **lsrole** command usage.

```
root@trinity:/root# lsrole -f storage_support
storage_support:
        authorizations=ibm.aix.management.cfgmgr,aix.device.manage.r
emove,aix.device.manage.create,aix.fs.manage.create,aix.fs.manage.re
move,aix.fs.manage.create,aix.fs.manage.mount,aix.fs.manage.unmount
        rolelist=
        groups=
        visibility=1
        screens=*
        msgcat=
        id=13

root@trinity:/root#
```

Figure 3-47   The lsrole command

4. Update the kernel security tables (KST) with the **setkst** command.

All RBAC security checking is performed in the AIX kernel. If the **setkst** command is not executed, the changes made to the RBAC security database will not be updated into the AIX kernel.

The **setkst** command is shown in Figure 3-48.

```
root@trinity:/root# setkst
Successfully updated the Kernel Authorization Table.
Successfully updated the Kernel Role Table.
Successfully updated the Kernel Command Table.
Successfully updated the Kernel Device Table.
root@trinity:/root#
```

Figure 3-48   Updating the Kernel Security Tables with the setkst command

5. Assign the user defined role storage_support to the storage support specialist user with the **chuser** command.

The storage support specialist has the AIX user name stor1.

In Figure 3-49, we list the roles (if any) assigned to the stor1 user.

```
root@trinity:/root# lsuser -a roles stor1
stor1 roles=
root@trinity:/root#
```

Figure 3-49   The lsuser command

By using the **lsuser** command, we can determine that the stor1 user has no roles assigned.

Use the **chuser** command to assign the storage_support user-defined role to the stor1 user.

Figure 3-50 shows the **chuser** command being used to assign the storage_support role to the stor1 user.

```
root@trinity:/root# chuser roles=storage_support stor1
root@trinity:/root# lsuser stor1
stor1 id=211 pgrp=storage groups=storage,staff home=/home/stor1
shell=/usr/bin/ksh gecos=John Smith login=true su=true rlogin=true
daemon=true admin=false sugroups=ALL admgroups= tpath=nosak ttys=ALL
expires=0 auth1=SYSTEM auth2=NONE umask=22 registry=files
SYSTEM=compat logintimes= loginretries=5 pwdwarntime=0
account_locked=false minage=1 maxage=13 maxexpired=-1 minalpha=1
minother=1 mindiff=1 maxrepeats=2 minlen=6 histexpire=4 histsize=4
pwdchecks= dictionlist= default_roles= fsize=2097151 cpu=-1
data=262144 stack=65536 core=2097151 rss=65536 nofiles=2000
roles=storage_support
root@trinity:/root#
```

Figure 3-50   Using the chuser command to assign a role

The stor1 user may now log in and use the **swrole** command to active the storage_support role.

By logging in as the stor1 user, we can test the role and privileged commands that we have assigned to the stor1 user.

Figure 3-51 shows the **lspv** command being executed by the stor1 user. Only one physical volume, hdisk0, is assigned to the system.

The stor1 user then activates the storage_support role with the **swrole** command.

```
stor1@trinity:/home/stor1# id
uid=211(stor1) gid=205(storage) groups=1(staff)
stor1@trinity:/home/stor1# lspv
hdisk0          00c4790e281766f5                    rootvg
active
stor1@trinity:/home/stor1# swrole storage_support
stor1's Password:
stor1@trinity:/home/stor1#
```

Figure 3-51   the stor1 user using the swrole command

After activating the storage_support role, the stor1 user may now execute the privileged commands that are assigned to the authorizations in the storage_support role.

The storage support specialist has assigned another disk to the system from the storage array and wishes to configure it. As the stor1 user has activated the storage_support role, the `cfgmgr` command can now be executed by the stor1 user.

Figure 3-52 shows the stor1 user executing the privileged `cfgmgr` command.

```
stor1@trinity:/home/stor1# lspv
hdisk0          00c4790e281766f5                           rootvg
active
stor1@trinity:/home/stor1# cfgmgr
stor1@trinity:/home/stor1# lspv
hdisk0          00c4790e281766f5 rootvg           active
hdisk1          00c4790e2ce9abc4 None
stor1@trinity:/home/stor1# id
uid=211(stor1) gid=205(storage) groups=1(staff)
```

Figure 3-52   The stor1 user executing the privileged command cfgmgr

The newly assigned disk is configured by the `cfgmgr` command and is defined as hdisk1.

By using Enhanced RBAC, we have successfully assigned a role to a general user that allows execution of a selected set of privileged commands.

We have complied with the least privileged principle and have not given the stor1 user access to any additional commands or files that are not required to complete the storage support tasks.

3.15.2 Scenario 2: Remote RBAC security database

In this scenario, we will configure the Enhanced RBAC security database to reside remotely on an LDAP server.

Prior to using LDAP as an RBAC database repository, the following prerequisites must be completed:

► An LDAP server must be configured and available to host the RBAC security database(s).

► The LDAP server must have the LDAP schema for RBAC installed. The RBAC schema is located in the /etc/security/ldap/sec.ldif file on the AIX V6 client.

► There must be network connectivity between the LDAP server and LDAP client.

► The LDAP client must have the LDAP client fileset installed.

► The LDAP client must be operating in Enhanced RBAC mode.

In this scenario, the administrator has chosen to host the RBAC security database on an LDAP server. The LDAP server has been previously configured by the LDAP administrator.

The administrator has chosen to host all five RBAC security databases on the LDAP server and use the LDAP server as the first search location for any RBAC database.

> **Important:** The procedure that follows outlines the steps to configure remote database hosting on an LDAP server for this environment. Your own environment may differ from the scenario environment, so consult your LDAP administrator before configuring the RBAC security database for remote LDAP hosting.

1. Export the RBAC security database.

   The RBAC security database must be exported in a format that will allow it to be updated into the LDAP server. By using the `rbactoldif` command, the RBAC security database can be exported to a file that will be used as an input for loading into the LDAP server with the `ldapadd` command.

   The location and name of the file used by the `rbactoldif` command may defined by the administrator. As with all system data files, the location of the file should not be in a public location. The file will be used only for the initial update of the LDAP database and may be removed once the initial LDPAP database update has been completed. Our administrator has chosen the root user's home directory and named the file /root/RBAC_DB.txt.

If only the selected RBAC security databases are chosen to be hosted on the LDAP server, then the **rbactoldif** command my be used with the -s option. The -s option allows for the selected databases to be hosted on LDAP. In this scenario, the administrator has chosen to use LDAP for all RBAC security databases.

Figure 3-53 shows the **rbactoldif** command. We have chosen the "cn" of aixdata. The output of the file is written to /root/RBAC_DB.txt.

```
root@trinity:/root# rbactoldif -d cn=aixdata > /root/RBAC_DB.txt
root@trinity:/root# more /root/RBAC_DB.txt
dn: ou=authorizations,cn=aixdata
objectclass: organizationalunit
ou:authorizations

dn: cn=Backup,ou=authorizations,cn=aixdata
cn: Backup
objectclass: ibm-authorization
ibm-authorizationid: 10001
```

Figure 3-53   Export RBAC security database with rbactoldif command

The RBAC security database output from the **rbactoldif** command can be viewed with a text viewer/editor. In Figure 3-53, we have used the **more** command to show the format of the data in the /root/RBAC_DB.txt file.

2. Update the RBAC security database to LDAP

The **ldapadd** command may be used to update the LDAP database with the data in the /root/RBAC_DB.txt file. The /root/RBAC_DB.txt file contains the information in the RBAC security database, that we exported in step 1 on page 242.

Before configuring the LDAP client, ensure that the LDAP server has the RBAC schema installed. The LDAP schema for RBAC is shipped with AIX V6 in the /etc/security/ldap/sec.ldif file. The LDAP administrator may need to update the schema on the LDAP server using the **ldapmodify** command.

In this scenario, we have used the **ldapadd** command with the -c option, which will cause the command to ignore an entry that may already exist and continue.

Figure 3-54 shows the **ldapadd** command being used to update the LDAP server with the RBAC security database information from the /root/RBAC_DB.txt file.

```
root@trinity:/root# ldapadd -h ldap -w admin -D cn=admin -c -i
/root/RBAC_DB.txt
root@trinity:/root#
```

Figure 3-54   The ldapadd command

The **ldapadd** command has now updated the LDAP server with the RBAC security database.

3. Configure the LDAP client.

The LDAP client fileset must first be installed, then the LDAP client may be configured with the **mksecldap** command.

> **Note:** The LDAP client fileset may depend on your specific AIX installation. At the time of publication, the LDAP client fileset used in this scenario is the 32-bit client, idsldap.clt32bit60.

The **mksecldap** command will configure the LDAP client and dynamically search the specified LDAP server to determine the location of the authorization, role, privileged command, device, and file data, and save the results to the /etc/security/ldap/ldap.cfg file.

Figure 3-55 shows the usage of the **mksecldap** command.

```
root@trinity:/root# mksecldap -c -h ldap -a cn=admin -p admin
root@trinity:/root#
```

Figure 3-55   The mksecldap command

> **Note:** The configuration variables used in Figure 3-55 are for this environment and may not match your own environment. Consult your LDAP administrator before configuring your LDAP client.

4. Update the RBAC security database domain lookup order.

After successfully configuring the system as an LDAP client through the **mksecldap** command, the client must be further configured to enable LDAP as a lookup domain for RBAC security database.

The /etc/nscontrol.conf file is used by RBAC to determine the lookup order for the RBAC security database domain.

By default, the RBAC security databases are located in the /etc/security/ directory. When the RBAC security databases are stored on an LDAP server, the /etc/nscontrol.conf file must be updated to include the LDAP domain for each RBAC security database stored on the LDAP server.

In this scenario, the administrator has chosen to have the five RBAC security databases located on the LDAP server, so the /etc/nscontrol.conf file must have the secorder stanza updated for the five RBAC security database.

Figure 3-56 shows the /etc/nscontrol.conf file prior to being configured to use the LDAP server as a lookup domain for the RBAC security database.

```
root@trinity:/etc/security/ldap#
authorizations:
        secorder = files
roles:
        secorder = files
privcmds:
        secorder = files
privdevs:
        secorder = files
privfiles:
        secorder = files
root@trinity:/etc/security/ldap#
```

Figure 3-56   the /etc/nscontrol.conf file, resolving local files only

The /etc/nscontrol.conf file may be updated using the `chsec` command to update the secorder stanza for each RBAC security database.

Figure 3-57 shows the `chsec` command, updating the secorder stanza of the /etc/nscontrol.conf file to set the LDAP domain as the first lookup domain.

```
root@trinity:/root# chsec -f /etc/nscontrol.conf -s authorizations
-a secorder=LDAP,files
root@trinity:/root# chsec -f /etc/nscontrol.conf -s roles -a
secorder=LDAP,files
root@trinity:/root# chsec -f /etc/nscontrol.conf -s privcmds -a
secorder=LDAP,files
root@trinity:/root# chsec -f /etc/nscontrol.conf -s privdevs -a
secorder=LDAP,files
root@trinity:/root# chsec -f /etc/nscontrol.conf -s privfiles -a
secorder=LDAP,files
root@trinity:/etc/security/ldap#
```

Figure 3-57   Using the chsec command to update the secorder stanza of /etc/nscontrol.conf

Figure 3-58 shows the /etc/nscontrol.conf file after being configured to use the LDAP server as a lookup domain for the RBAC security database.

```
root@trinity:/root#
authorizations:
        secorder = LDAP,files
roles:
        secorder = LDAP,files
privcmds:
        secorder = LDAP,files
privdevs:
        secorder = LDAP,files
privfiles:
        secorder = LDAP,files
root@trinity:/root#
```

Figure 3-58   The /etc/nscontrol.conf file, resolving first LDAP, then local files

5. Define the Kernel Security Table (KST) update method.

   By setting a value in the rbacinterval stanza in the /etc/security/ldap/ldap.cfg file, the /usr/sbin/secldapclntd daemon can be configured to automatically update the client KST.

By default in AIX V6, the rbacinterval stanza is set to 3600 seconds, but the stanza is commented out. The /usr/sbin/secldapclntd daemon also has a hard-coded value set of 3600 seconds. If no value is specified for the rbacinterval stanza in the /etc/security/ldap/ldap.cfg file, then the /usr/sbin/secldapclntd daemon will honor the 3600 value hard-coded in the daemon code.

If a value is specified in the rbacinterval stanza, then the rbacinterval stanza value will be used to determine the KST update frequency. An rbacinterval stanza value of 0 means that automatic update of the KST is disabled and any updates to the KST will need to be performed manually with the `setkst` command.

Figure 3-59 shows the default setting of the rbacinterval stanza in the /etc/security/ldap/ldap.cfg file.

```
# Time interval (seconds) for secldapclntd daemon to invoke setkst
# command to update the kernel RBAC tables. The value must be
greater
# than 60 seconds. Set to 0 to disable running of the setkst
command.
# The default value is 3600.
#rbacinterval:3600
```

Figure 3-59   The rbacinterval stanza in the /etc/security/ldap/ldap.cfg file

Note: In order to force an update of the KST from LDAP data, the `setkst` command can be executed from the command line.

6. Test the domain lookup with an RBAC database management command.

   By executing an RBAC database management command, the administrator can make sure that the domain lookup is functioning as expected.

   By using the `mkrole` command without specifying the -R option, the role will be defined in the first database that is defined by the secorder stanza in the /etc/nscontrol.conf file.

   In Figure 3-60 on page 248, we use the `mkrole` command to define a role with the name test_role. The secorder stanza defines the domain lookup as LDAP files, so the test_role role will be defined in the LDAP database.

Figure 3-60 shows the `mkrole` command.

```
root@trinity:/# mkrole dfltmsg='test role only' test_role
root@trinity:/# lsrole -R files test_role
3004-733 Role "test_role" does not exist.
root@trinity:/# lsrole test_role
test_role dfltmsg=test role only id=11
root@trinity:/#
```

Figure 3-60   Define a role in LDAP with the mkrole command

By using the `lsrole` command, we can see that the role has been defined in the LDAP database, and is not listed in the local /etc/security/roles file.

This shows that the RBAC security database domain lookup is working as defined in the /etc/nscontrol.conf field and LDAP client and server communications are active.

Any new RBAC definitions will be created in the LDAP database. If the administrator decides to create an RBAC definition in the RBAC security database in the local files, then the RBAC database management command lookup may be overruled by the use of the -R option.

Figure 3-61 shows the administrator using the `mkrole` command with the -R option to define the test_role_local in the local RBAC security database.

```
root@trinity:/# mkrole -R files dfltmsg='test role in local DB'
test_role_local
root@trinity:/# lsrole -R LDAP test_role_local
3004-733 Role "test_role_local" does not exist.
root@trinity:/# lsrole -R files test_role_local
test_role_local authorizations= rolelist= groups= visibility=1
screens=* dfltmsg=test role in local DB msgcat= id=12
root@trinity:/# lsrole test_role_local
test_role_local authorizations= rolelist= groups= visibility=1
screens=* dfltmsg=test role in local DB msgcat= id=12
root@trinity:/#
```

Figure 3-61   The lsrole command with the -R option

When the `lsrole` command is executed without a specific domain, the command returns the test_role_local information, because the domain search defines LDAP domain followed by the files domain in the search order.

When executing the `lsrole -R LDAP test_role_local` command, the search order is limited to the LDAP database only, where the role is not defined and cannot be listed.

The `lsrole -f files test_role_local` command lists the role from the local files database, and ignores the LDAP database search.

4

Trusted Execution environment

This chapter describes the Trusted Execution (TE) feature of AIX V6. This new component introduces a new command to verify a system's integrity while the Trusted Computing Base (TCB) is still available as an alternative. Unlike the TCB, which maintains checksums for crucial files and verifies them periodically (either triggered by `cron` or CLI), TE does such "offline checking" as well, but also allows for checking a file's integrity at its execution time, every time.

TE refers to a collection of features that are used to verify the integrity of the system's trusted computing base which in the context of TE is called *Trusted Signature Database* (TSD). In addition, TE implements advance security policies, which together can be used to enhance the trust level of the complete system.

The usual way for a malicious user to harm the system is to get access to the system and then install trojan horses, rootkits, or tamper with some security critical files such that the system becomes vulnerable/exploitable.

The central idea behind the set of features under TE is to be able to prevent such activities or in worst case be able to identify if any such thing happens to the system. Using the functionality provided by TE, the system administrator can decide upon the actual set of executables that are allowed to execute or the set of kernel extensions that are allowed to be loaded.

251

TE can also be used to audit the security state of the system and identify files that have changed, thereby increasing the trusted level of the system and making it more difficult for the malicious user to do harm to the system. The set of features under Trusted Execution (TE) can be grouped into:

► Managing the Trusted Signature Database

► Auditing integrity of the Trusted Signature Database (system integrity check)

► Configuring security policies (runtime integrity check)

► Trusted Execution Path, Trusted Library Path, Trusted Shell, and Secure Attention Key

Figure 4-1 shows Trusted Execution's two modes of operation.
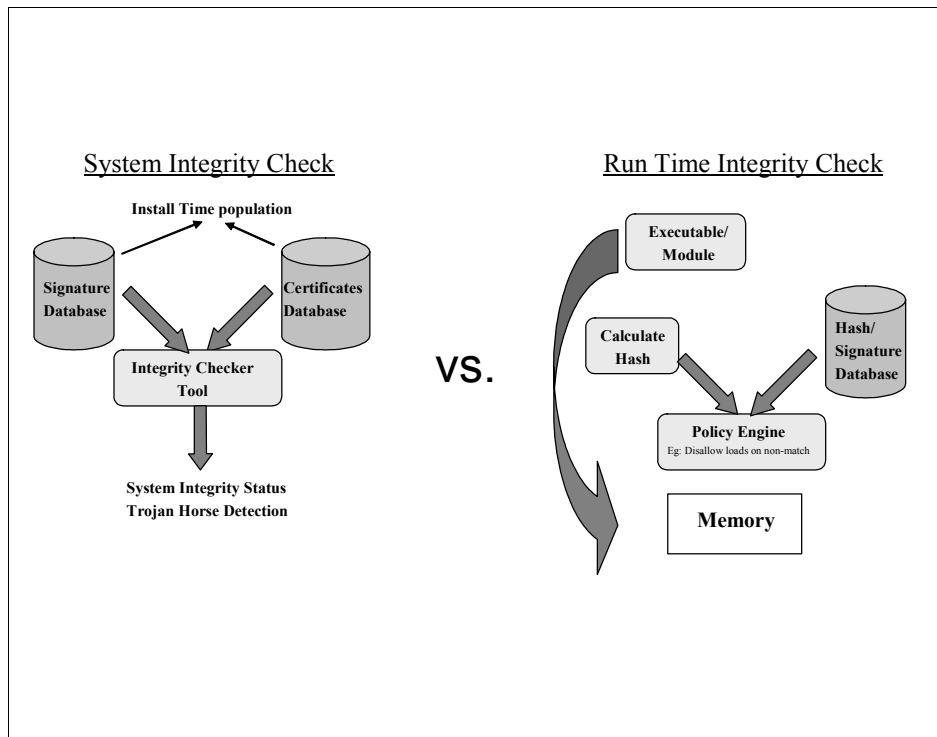
Figure 4-1   Trusted Execution's two modes of operation

In order for TE to work, the CryptoLight for C library (CLiC) and kernel extension need to be installed and loaded on your system. These filesets are included on the AIX Expansion Pack and are provided for free. To check whether they are installed on your system and loaded into the kernel, run:

```
# lslpp -l "clic*"
  Fileset                         Level  State       Description
  ----------------------------------------------------------------------------
Path: /usr/lib/objrepos
  clic.rte.includes            4.3.0.0  COMMITTED  CryptoLite for C Library
                                                   Include File
  clic.rte.kernext             4.3.0.0  COMMITTED  CryptoLite for C Kernel
  clic.rte.lib                 4.3.0.0  COMMITTED  CryptoLite for C Library

Path: /etc/objrepos
  clic.rte.kernext             4.3.0.0  COMMITTED  CryptoLite for C Kernel
# genkex|grep clic
        4562000     37748 /usr/lib/drivers/crypto/clickext
```

If the fileset is not installed, install it on your system through SMIT and load it into the kernel, once installation has completed successfully, by running:

```
# /usr/lib/methods/loadkclic
```

Note 1: Compared to TCB, TE is a more powerful and enhanced mechanism that overlaps some of the TCB functionality. While it could be used in addition on a TCB-enabled system, this would not result in a more secure system. It is a good and common practice to keep things small and simple (the KISS principle), hence you should choose one over the other.

Note 2: Trusted Execution is designed to run on a Trusted Software Stack (TSS) once the Trusted Platform Module[a] (TPM) becomes available for IBM System p servers. Since there are no such devices available at this point in time, TSS will not be enabled in AIX V6.

  a. Typically, the TPM is a cryptographic acceleration hardware component that is also used as a vault to store keys and other secrets. See also https://www.trustedcomputinggroup.org/groups/tpm/.

Note 3: Since Trusted Execution calculates cryptographic checksums on the fly, an impact on performance comes with it. However, the impact during testing for this book was not noticable. The overall impact heavily depends on the complexity of policies configured.

Still, once a binary has been verified before execution, it will remain in memory in case it gets called again. If the binary did not get paged out and has not been tampered with, consecutive executions will be allowed without calculating its hash again. This avoids any additional performance impact.

4.1 The Trusted Signature Database

Similar to that of TCB, there exists a database that is used to store critical security parameters of trusted files present on the system. This database resides at /etc/security/tsd/tsd.dat and comes with any AIX media. In TE's context, *trusted files* are files that are critical from the security perspective of a system and if compromised can jeopardize the security of the entire system. Typically the files that match this definition are:

► Kernel (operating system)

► All SUID root programs

► All SGID root programs

► Any program that is exclusively run by root or by a member of the system group

► Any program that must be run by the administrator while on the trusted communication path (for example, the `ls` command)

► The configuration files that control system operation

► Any program that is run with the privilege or access rights to alter the kernel or the system configuration files

Every trusted file should ideally have an associated stanza or a file definition stored in the TSD. A file can be marked as trusted by adding its definition in the TSD using the `trustchk` command. This command can be used to add/delete/list entries from the TSD. If desired, the TSD can be locked so even root cannot write to it any longer. (Locking the TSD becomes immediately effective. To unlock the TSD, a system reboot is required. The procedure of locking the TSD is described in detail in 4.3, "Configuring security policies (runtime integrity check)" on page 263.)

Every trusted file in the TSD is associated with a unique cryptographic hash and a digital signature. The cryptographic hash of the default set of trusted files is generated using the SHA-256 algorithm and the digital signature is generated using RSA by IBM AIX development and packaged as part of the AIX installation filesets. These hash values and signatures will be shipped as part of any AIX installation images and will be stored in the TSD on the destination system in a stanza format, as shown in Example 4-1.

Example 4-1   Example stanza of ksh command in the TSD

```
/usr/bin/ksh:
        owner = bin
        group = bin
        mode = TCB,555
        type = FILE
        hardlinks = /usr/bin/sh,/usr/bin/psh,/usr/bin/tsh,/usr/bin/rksh
        symlinks =
        size = 250820
        cert_tag = 00dcfcbcb7b0d911bd
        signature =
4009457bd5126b7ccc8962776e3aa81f0415fa507ed500dadab7c407ee29
56226542e1b6c3d0ed6a0691856bde1551f7f816f496a67e4faa72967f68de45801d
        hash_value =
54346893f1e5c193838c1673888421e2c78b12799e2e7b4ef46a3cdb200
31069
        minslabel =
        maxslabel =
        intlabel =
        accessauths =
        innateprivs =
        proxyprivs =
        authprivs =
        secflags =
```

Unneeded values in a stanza can either be set to NULL or completely skipped. All currently supported attributes are shown in Table 4-1.

Table 4-1   Attributes of the Trusted Signature Database

Attribute	Description
owner	Owner of the file. This value is computed by **trustchk** when the file is being added to the TSD.
group	Owner of the file. This value is computed by **trustchk** when the file is being added to the TSD.
mode	Comma separated list of values. This value is computed by **trustchk**. The permissible values are SUID, SGID, SVTX, and TCB. The file permissions must be the last value and can be specified as an octal value. For example, for a file that is set uid and has permission bits as rwxr-xr-x, the value for the mode shall be SUID,755.
type	Type of the file. This value is computed by **trustchk**. The possible values are FILE, DIRECTORY, MPX_DEV, CHAR_DEV, BLK_DEV, and FIFO.
hardlinks	List of hardlinks to the file. This value cannot be computed by **trustchk** and hence has to be supplied by the user while adding a file to the database.
symlinks	List of symlinks to the file. This value cannot be computed by **trustchk** and hence has to be supplied by the user while adding a file to the database.
size	Defines the size of the file. This value is computed by **trustchk**. A value of VOLATILE means the file gets changed frequently.
cert_tag	This value is computed by **trustchk** at the time of the addition of the file to the TSD. The field maps the digital signature of the file with the associated certificate that can be used to verify the file's signature. (Currently the certificate's ID is also its file name in /etc/security/certificates, but this might change in future releases.)
signature	The digital signature of the file. VOLATILE means the file gets changed frequently. This field is computed by **trustchk**.
hash_value	Cryptographic hash of the file. This value is computed by **trustchk**. VOLATILE means the file gets changed frequently.
minslabel	Defines the minimum Sensitivity Label for the object (when running Trusted AIX).
maxlabel	Defines the maximum Sensitivity Label for the object (when running Trusted AIX). This attribute is not applicable to regular files and FIFO.

Attribute	Description
intlabel	Defines the integrity label for the object (when running Trusted AIX).
accessauths	Defines the access authorization on the object (used in RBAC).
innateprivs	Defines the innate privileges for the file (used in RBAC).
inheritprivs	Defines the inherit privileges for the file (used in RBAC).
authprivs	Defines the privileges that will be assigned to the user if he or she has the given authorizations (used in RBAC).
secflags	Defines the file security flags associated with the object (used in RBAC). In addition to the values described in Chapter 3, "Role Based Access Control" on page 165, there is also the FSF_TLIB flag. It will mark the object as part of the Trusted Library.
t_accessauth	Defines the additional Trusted AIX specific access authorizations.
t_innateprivs	Defines the additional Trusted AIX specific innate privileges for the file.
t_authprivs	Defines the additional Trusted AIX specific privileges that will be assigned to the user if he or she has the given authorizations.
t_secflags	Defines the additional Trusted AIX specific file security flags associated with the object.

Adding files to the TSD is described in 4.5, "Signature creation and deployment" on page 268.

Note: Attributes of the TSD that are specific to Trusted AIX and RBAC are described in detail in Chapter 3, "Role Based Access Control" on page 165 and Chapter 5, "Trusted AIX/MLS" on page 273 respectively.

4.2 Auditing the integrity of the Trusted Signature Database (system integrity check)

The `trustchk` command can be used to audit the integrity state of the file definitions in the TSD against the actual files. This can be done by triggering `trustchk` periodically (either by `cron` or by CLI) just as you would have done with `tcbck`.

If `trustchk` identifies an anomaly, it can be instructed to auto-correct it or prompt the user before attempting any correction. In case of anomalies that are considered severe like size, signature, cert_tag, or hash_value mismatch, the correction is not possible since chances are likely that the original binary has been replaced.

The `trustchk` command would in such cases make the file inaccessible by removing all permissions from it. That way the file becomes useless and any damage arising from it is contained. Actions taken by `trustchk` depending on mismatched attributes are shown in Table 4-2.

Auditing with `trustchk` supports three options:

-n   Specifies the checking mode and indicates that errors are to be reported, but not fixed. Any discrepancy between attributes in TSD and actual file parameters will be reported by printing messages on stderr. the keyword "ALL" can be used to check all entries in the TSD.

-t   Specifies the checking mode and indicates that errors are to be reported with a prompt asking whether the error should be fixed. The keyword "ALL" can be used to check all entries in the TSD.

-y   Specifies the checking mode and indicates that errors are to be reported and fixed. This option should be used with care since it can potentially make a file unusable without user intervention if it encounters a discrepancy as described above. The keyword "ALL" can be used to check all entries in the TSD.

You can also validate file definitions against an alternate database using the -F option. This can be useful if you do not lock your local TSD or want to have one reference TSD for all your systems. That way you might store the TSD at some safe alternate location on a remote system and before running a periodic check the TSD file gets copied back first and then the check is run against it.

Table 4-2   Corrective actions taken by `trustchk`

Attribute	Corrective action
owner	The owner of the file will be reset to the value in the TSD.
group	The group of the file will be reset to the value in the TSD.
mode	The mode bits of the file will be reset to the value in the TSD.
type	The file will be made inaccessible.

Attribute	Corrective action
hardlinks	If the link points to some other file, it is modified to point to this file. If the link does not exist, a new link is created to point to this file.
symlinks	Same as hardlinks.
size	The file will be made inaccessible except in the case of a VOLATILE file.
cert_tag	The file will be made inaccessible.
signature	The file will be made inaccessible except in the case of a VOLATILE file.
hash_value	The file will be made inaccessible except in the case of a VOLATILE file.
minslabel	On a Trusted AIX system, the minimum Sensitivity Label will be reset to the value in the TSD.
maxlabel	On a Trusted AIX system, the maximum Sensitivity Label will be reset to the value in the TSD.
intlabel	On a Trusted AIX system, the integrity label will be reset to the value in the TSD.
accessauths	The access authorizations (RBAC) will be reset to the value in TSD. On Trusted AIX, the t_accessauths values are considered part of the accessauths attribute.
innateprivs	The innate privileges (RBAC) will be reset to the value in the TSD. On Trusted AIX, the t_innateprivs values are considered part of the innateprivs attribute.
inheritprivs	The inheritable privileges (RBAC) will be reset to the value in the TSD. On Trusted AIX, the t_inheritprivs values are considered part of the inherit attribute.
authprivs	The authorized privileges (RBAC) will be reset to the value in the TSD. On Trusted AIX, the t_authprivs values are considered part of the authprivs attribute.
secflags	The security flags (RBAC) will be reset to the value in the TSD. On Trusted AIX, the t_secflags values are considered part of the secflags attribute.

4.2.1 Examples of TE's auditing mode

The first example shows how **trustchk** works when it detects a binary that is still the original one but its file permissions and hardlinks have been changed:

```
# ls -laei telnet
   856 -r-xr-xr-x-    1 root     system       241568 May 09 18:27 telnet
# trustchk -y /usr/bin/telnet
Verification of attributes failed: mode
Verification of attributes failed: hardlinks
Verification of stanza failed: /usr/bin/telnet
# ls -laei telnet
   856 -r-sr-xr-x-    3 root     system       241568 May 09 18:27 telnet
```

Because those anomalies are not considered severe, the automatic, unprompted check simply reapplied the settings of the file as saved in the TSD.

In the next example, **trustchk** detects a severe anomaly, a change in size and hash value, as well as some other differences. This time the automatic, unprompted check renders the file inaccessible by removing its permissions:

```
# trustchk -y /usr/bin/telnet
Verification of attributes failed: mode
Verification of attributes failed: hardlinks
Verification of attributes failed: size
Verification of attributes failed: hash
Verification of stanza failed: /usr/bin/telnet
# ls -laei /usr/bin/telnet
 2456 ---------T- 1 root system 241570 May 09 18:30 /usr/bin/telnet
```

4.2.2 Checking the signing authority

To answer the legitimate question "who has actually signed this binary?", you simply need to take a quick look at the TSD. First, you discover the certificate ID the binary was signed with:

```
# trustchk -q /usr/sbin/telnetd|grep cert_tag
        cert_tag = 00dcfcbcb7b0d911bd
```

Next, check if a file with that name exists in /etc/security/certificates and run this command against that file:

```
# openssl x509 -inform DER -in 00dcfcbcb7b0d911bd -text -noout
```

The output might look similar to this:

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            dc:fc:bc:b7:b0:d9:11:bd
        Signature Algorithm: sha1WithRSAEncryption
        Issuer: C=US, ST=TX, L=AU, O=IBM Corporation, OU=AIX,
CN=www.ibm.com
        Validity
            Not Before: Jan 25 06:33:57 2007 GMT
            Not After : Jan 24 06:33:57 2010 GMT
        Subject: C=US, ST=TX, L=AU, O=IBM Corporation, OU=AIX,
CN=www.ibm.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (512 bit)
                Modulus (512 bit):
                    00:ea:26:f5:08:c0:d4:14:29:91:31:ee:d7:6c:e5:
                    8c:71:a1:a8:d2:28:5d:ba:fd:f2:04:7d:b5:58:29:
                    c7:72:ea:64:0b:c6:7f:a8:f8:f0:75:c7:8f:1e:1a:
                    cc:f2:d1:fb:69:45:19:38:45:88:fc:38:5e:ea:3e:
                    d2:64:36:cb:63
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Subject Key Identifier:

F8:CE:2E:C3:D3:03:A3:B0:85:8D:1D:40:4B:82:A9:62:80:E9:01:35
            X509v3 Authority Key Identifier:

keyid:F8:CE:2E:C3:D3:03:A3:B0:85:8D:1D:40:4B:82:A9:62:80:E9:01:35
                DirName:/C=US/ST=TX/L=AU/O=IBM
Corporation/OU=AIX/CN=www.ibm.com
                serial:DC:FC:BC:B7:B0:D9:11:BD

            X509v3 Basic Constraints:
                CA:TRUE
    Signature Algorithm: sha1WithRSAEncryption
        79:c5:6c:dd:c5:8a:bb:1e:4d:71:1e:24:96:d2:96:a5:8a:86:
        8a:ab:cd:7a:58:a2:2d:50:b2:8a:f1:c2:e9:81:81:9c:fe:c0:
        90:5f:3d:57:fc:31:18:d4:11:6e:41:a9:b4:31:5f:35:06:ad:
        92:07:ce:45:f0:31:7d:00:3c:7b
```

If you cannot find a matching file name, look at the default certificate called /etc/security/certificates/certificate_540. The matching certificate has to be in /etc/security/certificates, whatever its name might be. It is actually the certificate's ID that gets used by `trustchk`, but this ID does not necessarily have to be the certificate's file name as well. The TSD attribute cert_tag is in fact the certificate's serial number without colons and maybe some padding zeros at the beginning. If no matching certificate is found at all on the system, integrity checking will fail.

4.3 Configuring security policies (runtime integrity check)

In addition to the system integrity check (described in the previous chapter), which is very similar to the one offered by `tcbck` on a TCB-enabled system, TE also provides you with a runtime file integrity verification mechanism. (Keep in mind that TE is not an install-time-only option like TCB is, so you can use this feature anytime.)

This mechanism checks the integrity of the trusted files before every request to access those file, effectively allowing only the trusted files that pass the integrity check to be accessed on the system.

When a file is marked as trusted (by adding its definition to the TSD), the TE feature can be used to monitor its integrity on every access. TE can continuously monitor the system and hence is capable of detecting tampering of any trusted file (by a malicious user or application) present on the system at runtime (that is, at load time). If the file has been tampered with, TE can take corrective actions based on pre-configured policies. If a file is being opened or executed and has an entry in the TSD, TE will behave as follows:

1. Before loading the binary, the component responsible for loading the file (system loader) will invoke the TE subsystem, which will calculate the hash value using the SHA-256 algorithm.

2. This runtime calculated hash value will be matched with the one stored in the TSD.

3. Only if the values match, the file opening/execution will be allowed.

4. If the values do not match, because either the binary is tampered or somehow compromised, the policy configured by you defines the relevant actions. Available policies are shown in Table 4-3.

Table 4-3   Available policies for runtime integrity checking

Policy	Action
CHKEXEC	Checks the integrity of trusted executables before loading them in memory for execution.
CHKSHLIB	Checks the integrity of trusted shared libraries before loading them in memory for execution.
CHKSCRIPT	Checks the integrity of trusted shell scripts before loading them in memory.
CHKKERNEXT	Checks the integrity of kernel extensions before loading it in memory.
STOP_UNTRUSTD	Stops loading of files that are not trusted, that is, only files belonging to the TSD will be loaded. This policy works in combination with any of the CHK* policies mentioned above. For example, if CHKEXEC=ON and STOP_UNTRUSTD=ON, then any executable binary that does not belong to the TSD will be blocked from execution.
STOP_ON_CHKFAIL	Stops loading of trusted files that fail the integrity check. This policy also works in combination with CHK* policies. For example, if CHKSHLIB=ON and STOP_ON_CHKFAIL=ON, then any shared library that does not belong to the TSD will be blocked from getting loaded into memory for use.
TSD_LOCK	Lock the TSD so that it is no longer available for modification through the `trustchk` command. Enabling this policy immediately locks the TSD; disabling it requires a reboot of the system (as any change to an active policy).
TSD_FILES_LOCK	Lock trusted files. No change to any TSD files is allowed.
TEP	Sets the value of the Trusted Execution Path, and enables or disables it. The TEP consists of a list of colon separated absolute paths like /usr/bin:/usr/sbin. When this policy is enabled, the files belonging to only these directory paths are allowed to be executed. Any executable program requested to be loaded that does not belong to the TEP will be blocked.

Policy	Action
TLP	Sets the value of the Trusted Library Path, and enables or disables it. The TLP consists of a list of colon separated absolute paths like /usr/lib:/usr/ccs/lib. When this policy is enabled, the libraries belonging to only these directory paths are allowed to be loaded. Any program trying to load a library that does not belong to the TLP will be blocked.
TE	Enable/Disable Trusted Execution functionality. Only when this policy is enabled does the above mentioned policies come into effect. Table 4-4 on page 266 shows the interaction of different CHK* policies with STOP* policies when enabled.

Note: A policy can be enabled or disabled at any time until TE is turned ON to bring the policies into effect. Once a policy is in effect, then disabling that policy will go into effect only on the next boot cycle. All the information messages are logged to `syslogd`.

Here are some examples for **trustchk** in runtime integrity mode:

To enable TSD protection, run:

```
# trustchk -p tsd_lock=on
# trustchk -p te=on
```

The TSD is immediately protected against any kind of modification then. Neither **trustchk** nor a manual edit of the file is possible:

```
# trustchk -d /usr/bin/ps
Error writing to database file
# echo >> /etc/security/tsd/tsd.dat
Operation not permitted.
```

To enable the TSD for write access again, you either need to switch off TE completely or set tsd_lock to off. Either way, you need to reboot in order to have this change become active immediately:

```
# trustchk -p te=off
Policy in use. Changes applicable on next boot only
```

When blocking any untrusted shell scripts by using the CHKSCRIPT policy, make sure all scripts needed by your services are included in the TSD. For example, if you are using OpenSSH, make sure the Ssshd and Ksshd start and stop scripts in /etc/rc.d/rc2.d are in the TSD. Otherwise, `sshd` will not get started upon reboot and not be shut down on a system `shutdown`:

```
# trustchk -p stop_untrustd=on
# trustchk -p chkscript=on
```

When trying to start a script with chkscript=on and that script is not included in the TSD, its execution will be denied, regardless of its permissions, even when root is invoking it:

```
# ./foo
ksh: ./foo: 0403-006 Execute permission denied.
# ls -l foo
-rwx------- root     system           17 May 10 11:51 foo
```

In addition, TE will write a log message to `syslogd`, for example:

```
kern:info unix: Trusted Execution: Crypto hash verification failed:
./foo
```

Table 4-4 shows the interaction of CHK* with the STOP* policies.

Table 4-4   Interaction of CHK* with STOP* policies

CHK* Policy	STOP_UNTRUSTD	STOP_ON_CHKFAIL
CHKEXEC	Stops loading of executables that do not belong to the TSD	Stops loading of executables whose hash values do not match the TSD values
CHKSHLIB	Stops loading of shared libraries that do not belong to the TSD	Stops loading of shared libraries whose hash values do not match with the TSD values
CHKSCRIPT	Stops loading of shell scripts that do not belong to the TSD	Stop loading of shell scripts whose hash values do not match with the TSD values
CHKKERNEXT	Stops loading of kernel extensions that do not belong to the TSD	Stop loading of kernel extensions whose hash values do not match with the TSD values

4.4 Trusted Execution Path, Trusted Library Path, Trusted Shell, and Secure Attention Key

The Trusted Execution Path (TEP) defines a list of directories that contain the trusted executables. Once TEP verification is enabled, the system loader allows only binaries in the specified paths to execute. For example:

```
# trustchk -p tep
TEP=OFF
TEP=/usr/bin:/usr/sbin
# trustchk -p
tep=/usr/bin:/usr/sbin:/etc:/bin:/sbin:/usr/lib/instl:/usr/ccs/bin
# trustchk -p tep
TEP=OFF
TEP=/usr/bin:/usr/sbin:/etc:/bin:/sbin:/usr/lib/instl:/usr/ccs/bin
# trustchk -p tep=on
# trustchk -p tep
TEP=ON
```

The Trusted Library Path (TLP) has the same functionality as TEP with the only difference that it is used to define the directories that contain trusted libraries of the system. Once TLP is enabled, the system loader will allow only the libraries from this path to be linked to the binaries. The `trustchk` command can be used to enable/disable the TEP/TLP as well as to set the colon-separated path list for both using TEP and TLP command-line attributes of `trustchk`:

```
# trustchk -p tlp
TLP=OFF
TLP=/usr/lib:/usr/ccs/lib:/lib:/var/lib
```

As already mentioned in Table 4-1 on page 257, TLP uses a flag to control its operations: FSF_TLIB. If the file has the FSF_TLIB flag set in its TSD stanza, then the process resulting from it will be set as a TLIB process. Processes marked as TLIB processes can link only to *.so libraries that also have the TLIB flag set.

> **Note:** Be careful when changing either TEP or TLP. We do not recommend removing paths from their default settings, which are currently set to:
>
> ```
> TEP=/usr/bin:/usr/sbin:/etc:/bin:/sbin:/sbin/helpers/jfs2:/usr/lib/i
> nstl:/usr/ccs/bin
> TLP=/usr/lib:/usr/ccs/lib:/lib:/var/lib
> ```
>
> Doing so will most probably result in a system that will not reboot and function properly since it cannot access necessary files and data any longer.

The Trusted Shell (tsh) and the Secure Attention Key (SAK) behave in a similar way to that of the Trusted Computing Base (TCB) with the only difference that if TE is enabled on the system instead of TCB, tsh would execute files belonging to TSD only. For more details, such as enabling SAK, and so on, please refer to the SAK details in TCB documents for this or earlier AIX releases[1].

4.5 Signature creation and deployment

In order for all checks and policies of TE to be trustworthy, signatures are provided by IBM (or the respective ISV for any other product). In case signatures are missing due to whatever circumstances, such files can be added to the TSD in two ways:

1. The signature creation and deployment process, as conducted by IBM, is designed as shown in Figure 4-2 on page 269.

2. You create your own binaries and include them into the TE framework.

4.5.1 Adding BFF files to the TSD

When a fileset gets assembled into Backup File Format (BFF), the hash values for the TSD get created using IBM keys and certificates.This is done with the help of development tools like `builtsecattr` and `instsecattr`. Only the latter one will be shipped with the AIX media. The `instsecattr` command is called by `installp` and `geninstall` to add a trusted file's stanza to the TSD in order to guarantee the integrity of these files from IBM install media. If you do not use BFF files, then `trustchk` will be used to add a trusted file's stanza to the TSD.

---

[1] http://publib.boulder.ibm.com/infocenter/pseries/v5r3/index.jsp?topic=/com.ibm.aix.security/doc/security/tcb_configuring_additional_config_sak.htm
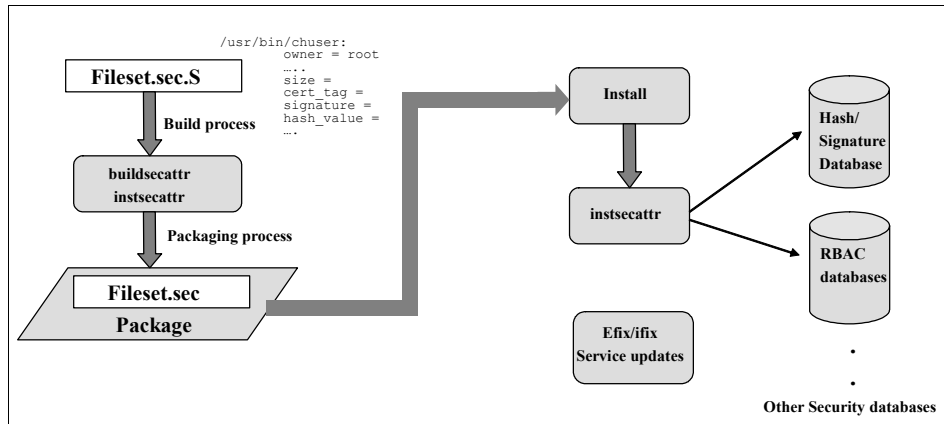
Figure 4-2   Trusted Execution signature creation and deployment at IBM

4.5.2 Adding non-BFF files to the TSD

If you want to add your own (or ISV) binaries to the TSD, only **trustchk** and **openssl** are needed. While **trustchk** is available on any AIX V6 system anyway, **openssl** can be used on one dedicated system only to set up your private key and certificate to be able to compute checksums for the TSD. This key and its corresponding certificate need to be copied to all systems where you want to add files to the TSD, unless you create the files' stanzas on your development system and copy the stanzas to your reference image or your other systems.

Make sure OpenSSL is installed on your AIX system. At least Version openssl-0.9.7l-1.aix5.1.ppc.rpm needs to be installed, which can be found on the AIX Toolbox for Linux Applications CD or on the respective IBM Web site, the AIX Toolbox Cryptographic Content page[2].

To check whether the proper version of OpenSSL is installed on your system, run:

```
# lslpp -L "openssl*"
  Fileset                      Level  State  Type  Description (Uninstaller)
  ----------------------------------------------------------------------------
  openssl                      0.9.7l-1   C     R    Secure Sockets Layer and
                                                     cryptography libraries and
                                                     tools (/bin/rpm)
```

---

2 http://www6.software.ibm.com/dl/aixtbx/aixtbx-p

If OpenSSL is not installed on your system or an older version is installed, install the required version either from CD or from a file system where you have downloaded the RPM file to:

```
# rpm -i openssl-0.9.7l-1.aix5.1.ppc.rpm
```

The setup for a signed private/public keypair is much simpler than the one described in 6.8, "Secure File Transfer Protocol" on page 364 for secure ftp setup because **trustchk** does not need a complete chain of trust up to some root Certificate Authority. Since **openssl** creates its keys and certificates in privacy enhanced mail security certificate (PEM) format, they need to converted into ASN.1/PKCS8/DER (distinguished encoding rules) format to become usable for TE. This can be quickly done by using the following commands:

```
# openssl genrsa -out privkey.pem 2048
```

Which will generate a 2048-bit private key in PEM format.

```
# openssl req -new -x509 -key privkey.pem -outform DER -out cert.der
-days 3650
```

Which will create the public key and certificate that lasts approximately ten years:

```
# openssl pkcs8 -inform PEM -in privkey.pem -topk8 -nocrypt -outform
DER -out privkey.der
```

Which will convert the private key from PEM into DER format. After the conversion, the private key in PEM format is no longer needed. You only need privkey.der and cert.der for the following examples. Now you can add any file you would like to the TSD by simply issuing:

```
# trustchk -s privkey.der -v cert.der -a /path/to/binary
```

Unless when adding a VOLATILE file, a directory, or a device file, it is necessary to provide a private key during the addition of file definitions to the TSD using the -s flag and an associated digital certificate carrying the associated public key using the -v flag. The private key is used to generate the signature of the file and subsequently it will be discarded, so it is up to you to store this key securely. (At the time of writing, the certificate is stored in /etc/security/certificates and will be used to verify the signatures whenever integrity verification is requested by you. This might change in future releases of AIX.)

A direct update of an existing stanza for a binary that was modified deliberately (that is, fixed) is not possible. The stanza needs to be deleted first and then re-added:

```
# trustchk -d /path/to/binary
# trustchk -s privkey.der -v cert.der -a /path/to/binary
```

When you are adding a new entry to the TSD and that trusted file has some symbolic or hard links pointing to it, these links must be manually added to the TSD by using the symlinks and hardlinks attributes at the command line along with **trustchk**. Here is an example:

```
# trustchk -s key -v cert -a /usr/bin/ksh
hardlinks=/usr/bin/sh,/usr/bin/psh,/usr/bin/tsh,/usr/bin/rksh
```

If the file being added is expected to be changing very frequently, then this can be indicated to the **trustchk** command using `size=VOLATILE` on the command line. This will instruct **trustchk** not to calculate the hash_value and signature fields while generating the file definition for adding to the TSD. During integrity verification of this file, the hash_value and signature fields will be ignored.

You can also supply any pre-computed file definitions (for example, from your development system, as discussed in this section) through a file using the -f option to be added to the TSD. In this case, **trustchk** will not compute any of the values and will store the definitions into the TSD without any verification. When doing so, the user is responsible for the sanity of the file definitions. Private key and public certificate are not needed when using a pre-computed stanza file.

For a complete description of all options of **trustchk**, please refer to Table 4-1 on page 257.

> **Note:** As shown in Figure 4-2 on page 269, iFixes are not provided with hash values by IBM. It is your responsibility to add the files included in such fixes to the TSD if needed.

5

Trusted AIX/MLS

In this chapter, we cover the Multi-level Security feature introduced in Trusted AIX. We will discuss the new security attributes added as part of the Multi-level Security and show how they enhance the security of the information and make the AIX a *Trusted Operating System*. This chapter also covers how do we install, configure, and manage a trusted AIX system and what kind of installation options are available. At the end of the chapter, we will show how to configure Trusted AIX and how commercial applications can be set up to run successfully on Trusted AIX.

5.1 Overview

This section gives an overview of what is Multi Level Security, what is the need of enhanced security, and how does Trusted AIX provides the enhanced security. We also talk about some standards for Multi Level Security.

5.1.1 What is Multi Level Security

Multi Level Security is about classifying information at various level and decide the access policy based on their security level. In Trusted AIX, Multi Level Security is based on labelling the information with different labels and controlling the access based on the labels. As a Multi Level Security feature, Trusted AIX introduces two more access controls over the traditional Discretionary Access Control that is based on the discretion of the owner of the object. We discuss these additional security layers, in detail, later in this chapter. These access controls are enforced by the system and not by the user, which makes it more secure. The additional security layers over Discretionary Access Control (DAC) are known as:

► Mandatory Access Control (MAC)
► Mandatory Integrity Control (MIC)

5.1.2 What is the need for enhanced security

Today's world is an information oriented world. The information manager finds it difficult to secure the highly important corporate data from unauthorized users. They need some means that is easy to maintain and protects their data

Traditional UNIX cannot completely provide such security because the user root (the superuser of the system) can either in authorized or unauthorized manner can do anything with the system and nothing in the system stops them from doing so.

The threat involved is not only external but also internal; the information can be leaked by the malicious insiders. Either way, state-of-the-art security systems used by the financial industry have not been designed to prevent the transmission of sensitive information within an organization.

It is possible that anyone in an organization, from low-level clerks to senior vice presidents and other corporate officers, could transmit sensitive data outside of these institution's networks without detection by firewalls or other network traffic monitoring systems designed to prevent this from happening.

5.1.3 What Trusted AIX provides

Trusted AIX provides the *Multi Level Security* capability that is enforced by the system and not by the owner of the system resources. A properly configured system is resistant to attacks from the inside as well as outside of the system. The system manages all the access to the data. Users who need to modify system configuration parameters will have to be assigned sufficient authorizations to be able to successfully execute the relevant commands. A user cannot do any thing more than for which he is authorized. The system uses assigned policies to control access.

Policy decision includes classifying the information, deciding the labels and their hierarchy to be used, labelling the information with the labels, and enabling or disabling various label checks.

The information is labeled in different levels and every one from high rank post to low rank post has restricted access to the information based on labels assigned to them in Trusted AIX. We will see in our hypothetical scenario how Trusted AIX solves the problem of internal leakage, and how the information is classified and saved from leakage.

The following are the four primary element of information security which Trusted AIX addresses.

► Confidentiality

  Threats related to disclosure of the information to the unauthorized users are a confidentiality issue.

  Trusted AIX provides an Object Reuse and Access control mechanism to prohibit information leakage through system resources. Resources could be internal to the system, such as buffers and caches, as well as external, such as printer, disk, and tap drives. Object reuse refers to the allocation and deallocation of system resources (storage objects) to the subject. The security of the system requires that no system resources can be used to transfer data from one process to another and from one user to another.

  Trusted AIX provides Labeled Printing. Each print job is automatically provided a banner page and a trailer page that shows all security relevant labels. When writing data to disks and tape in AIX with the `backup` command, Sensitivity Labels are included with the data. Only an authorized user can import or export unlabeled data from tapes or disks.

▶ Integrity

Related to modification of the information by an unauthorized person.

Trusted AIX provides access control policy such that unauthorized users can not modify the information, whether it is information generated inside the system or coming from outside of the system. To prevent the modification of the information by a malicious user, root privilege has been disabled in the system.

▶ Availability

Related to the availability of the system resource or services to the authorized users.

Trusted AIX protects the system by unauthorized users and processes that create denial of services. If a malicious program fills up the disk space by writing garbage data, the other users will not be able to create their files. In Trusted AIX, unprivileged process are not allowed to create or read the files in a directory.

▶ Accountability

Related to accounting which process or user performed what action.

The Audit subsystem records all security related events. The audit services provide the administrator a set of auditable events and an audit trail of all security related system events.

5.1.4 Historical aspect

Many standards have been published for the security of classified labeled information.

The United State Government Department of Defense (DoD) published the Trusted Computer System Evaluation Criteria (TCSEC) in December 1985. This states the security requirements for the Automatic Data Processing systems. This standard is used for assessing the effectiveness of the Computer Security controls built in to a Computer system. The TCSEC was used to evaluate, classify, and select computer systems being considered for the processing, storage, and retrieval of sensitive or classified information.

TCSEC was then replaced by the Common Criteria International Standard originally published in 2005.

Trusted AIX Multi Level Security has been designed keeping these standards in mind.

5.2 Introduction to MLS

We shall use a bookshelf metaphor to describe MLS. Books you do not want to be accessed by everyone will be put in the topmost shelf and the ones you want everyone to have access to will be placed in the lowest shelf, assuming that height is our security criteria (see Figure 5-1).
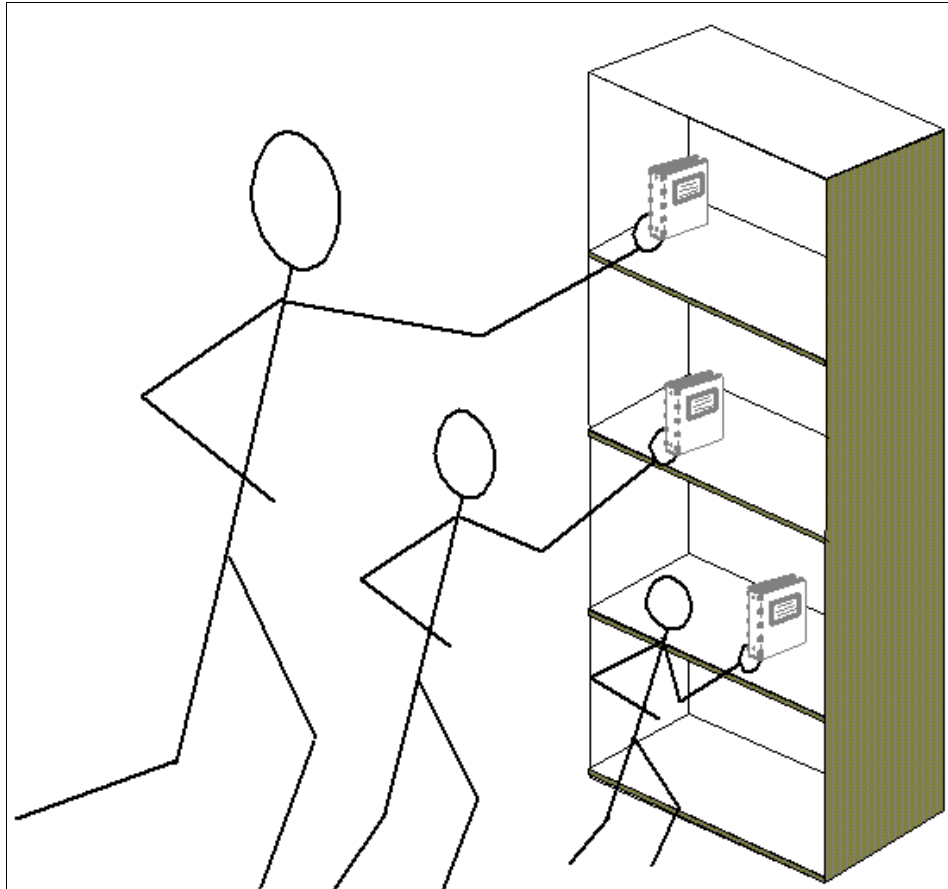
Figure 5-1   People accessing the information at different level base on height

In exceptional cases where we need to provide access to a person who is at lower level, users can be given privileges to cross these security checks, such as when a user is running in lower level and runs an executable that needs to cross these label checks. Trusted AIX provides the functionality to make it succeed using *privileges* (see "Privilege" on page 292 and Figure 5-2).
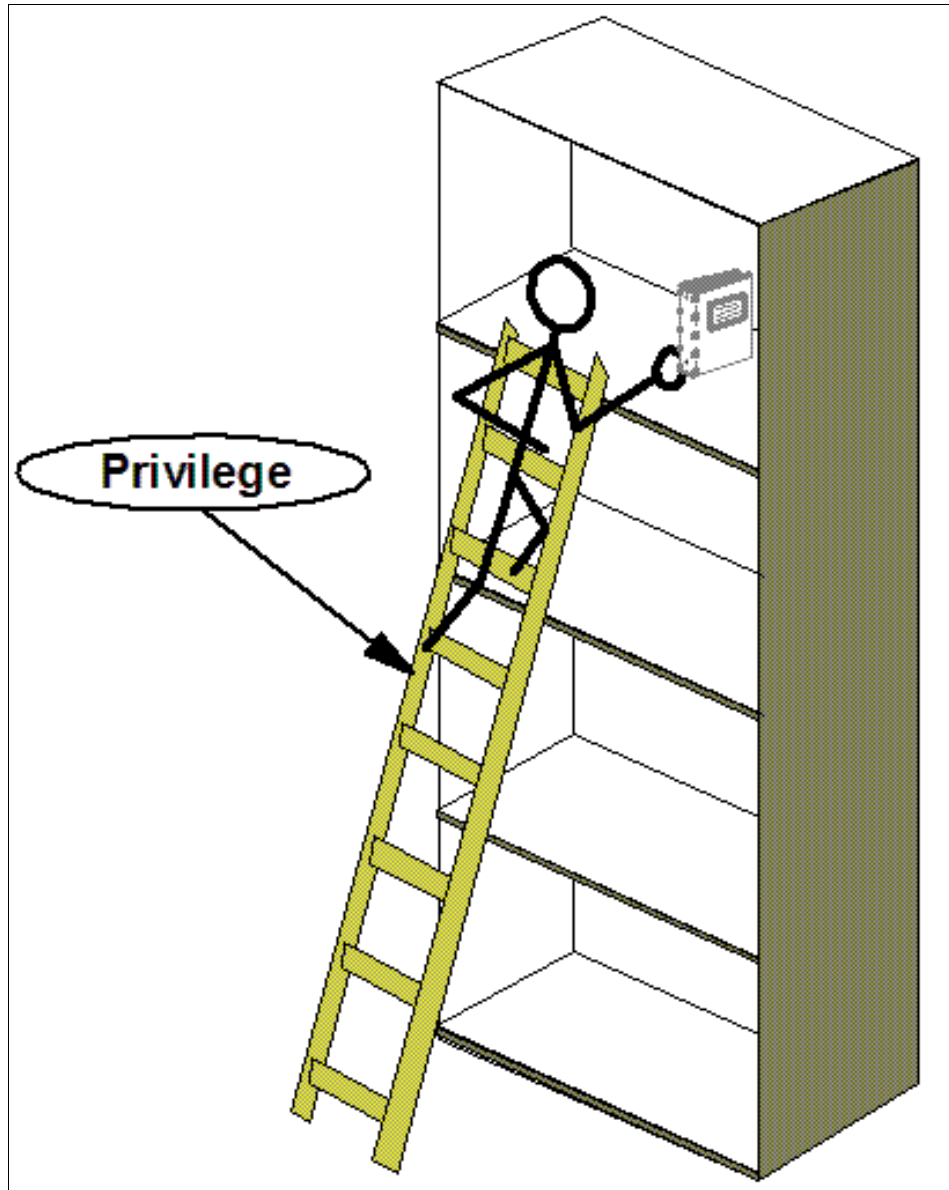
Figure 5-2   Privilege to cross the security check

5.2.1 What is new in Trusted AIX

Security features provided by a standard UNIX system include:

► A login password protected system and network access

► User group and world file access permission

► Access control list

► Audit subsystem

All the above attributes of the objects provided by the traditional UNIX system are based on the discretion of the owner of that object; this is called *Discretionary Access Control*. The owner may change the security information of the object he owns, willingly or unwillingly. That could cause the leak of information to other users who are not supposed to have that information.

In addition to these security attributes, Trusted AIX provides the security attributes that are controlled not by the owner but by the system and the administrator of the system. So a user with malicious intentions still cannot leak the information by changing the security attributes of the object.

Trusted AIX provides access control based on the sensitivity and integrity of the information. The access control based on the sensitivity of the information, object holds, and the sensitivity clearance of the process or user is called *Mandatory Access Control*. Such access is enforced by the system and not the owner of the object or resource.

Access control based on maintaining the integrity of the information and verifying the trustworthiness of the user is called *Mandatory Integrity Control*.

These three level of access controls can be viewed as shown in Figure 5-3. MAC and MIC make two more layers of security around DAC to make the system security better.
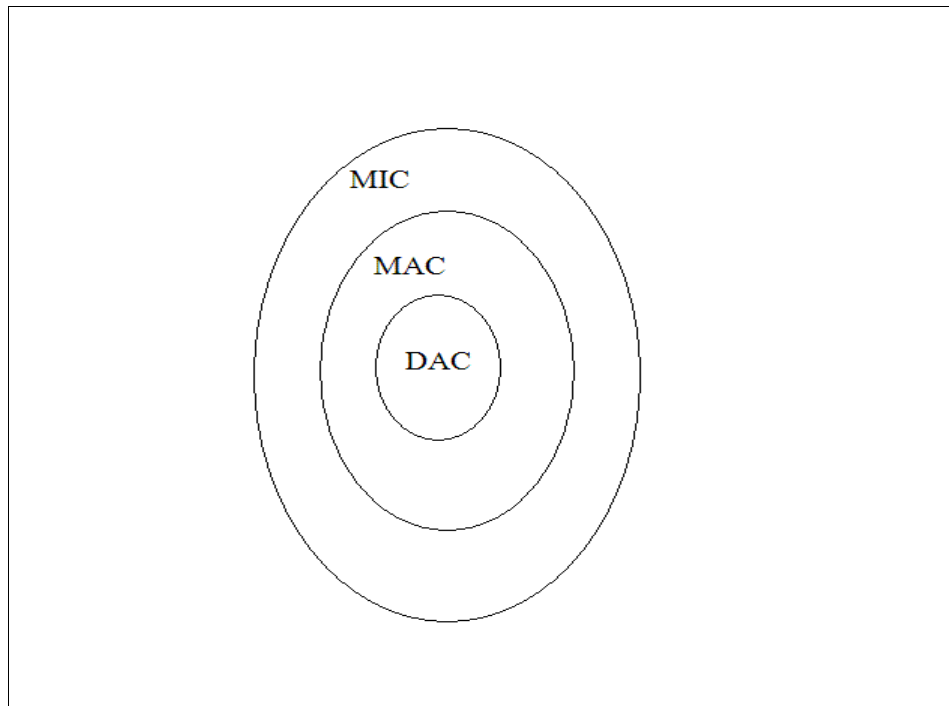
Figure 5-3   View of MAC, MIC, and DAC

The following are the new Security attributes introduced as part of the security enhancement added in Trusted AIX.

► Sensitivity Label

► Integrity Label

► File Security Flags

► Kernel Security Flags

In addition to these new attributes, two very important attributes for Trusted AIX were introduced as part of the RBAC, and Trusted AIX adds more privileges and authorizations to the system. They are:

► Privileges

► Authorization

All the file system entities and the resources are called objects and all the other entities that use the resources are called subjects. For example files, directories and IPC objects are called objects and the processes are called subjects.

5.2.2 Mandatory Access Control

Mandatory Access Control is based on the sensitivity of the information an object holds and the sensitivity clearance of the subject. MAC is controlled by the system. The system enforces access based on the level of sensitivity. In Trusted AIX, the sensitivity of the information is measured by labels assigned to the object or subject; this is called a *Sensitivity Label* (SL).

All the objects and the subjects in the system are assigned Sensitivity Levels and the system compares the subject Sensitivity Label and object Sensitivity Label to determine access.

See Figure 5-4 for more information.
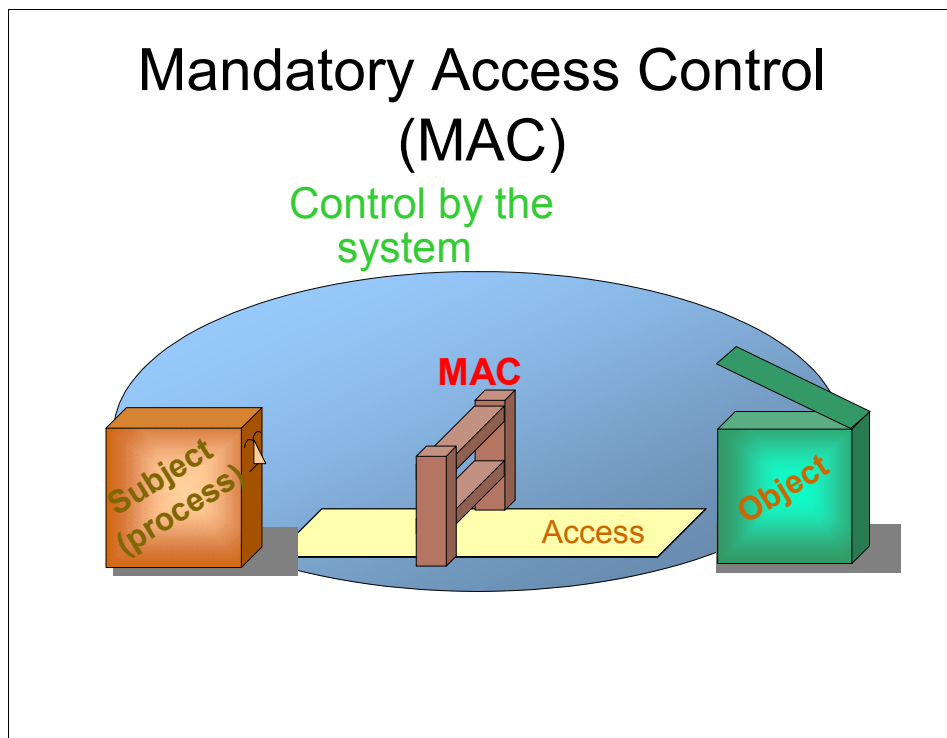
Figure 5-4   Mandatory Access Control

What is a Sensitivity Label

A Sensitivity Label has two parts:

- ▶ Classification
- ▶ Compartments

Classification

Classification indicates the level of security. These are system defined hierarchical labels. The maximum number of classifications possible in the system is 32,000.

Trusted AIX comes with some default classification labels, that is, TOP SECRET, SECRET, and CONFIDENTIAL (see Figure 5-5).
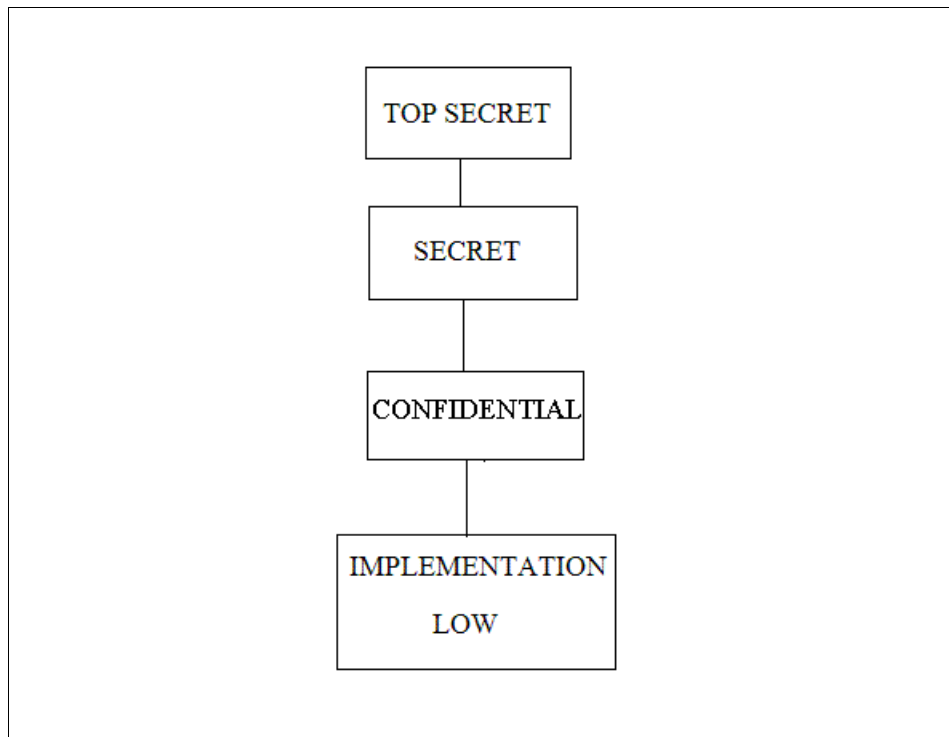
Figure 5-5   HIerarchy of classification

Compartments

We can have different departments or groups where we set the different levels of security in the department as well as inter department or intergroup. Compartments can be used for representing different departments and groups.

Compartments are represented as strings. Strings can be *any* name.
Compartments do not have any intrinsic ordering, but the administrator can
impose constraints on which compartments and classifications can be combined.

Figure 5-6 shows some examples of compartments.
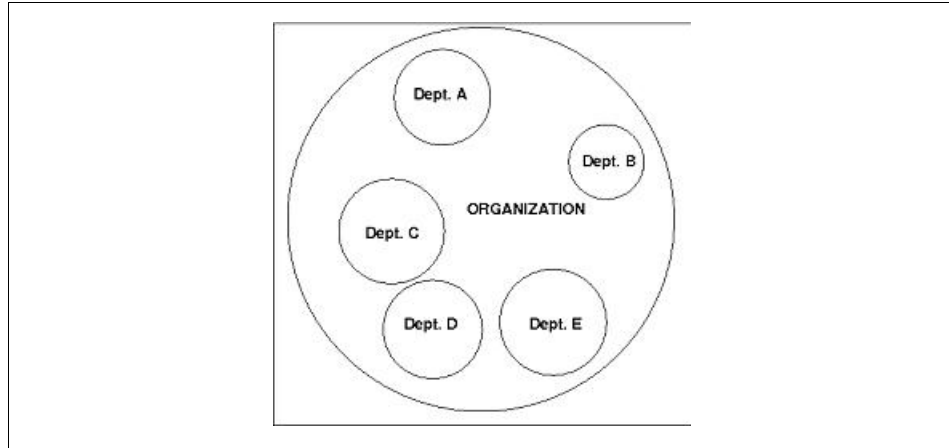
Figure 5-6   Example of compartments

A label is formed by the combination of a classification and the compartments, as shown in Figure 5-7.
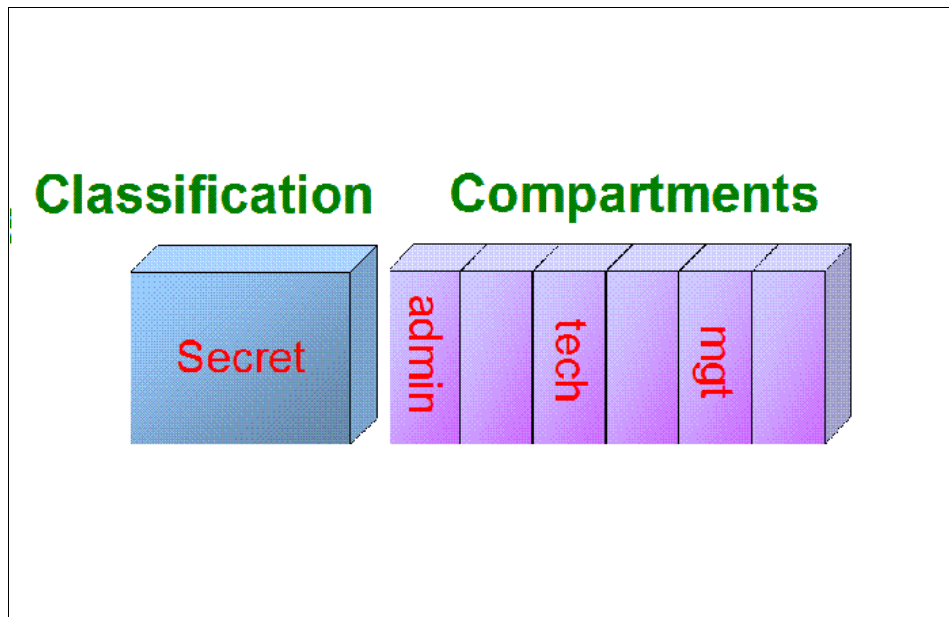
Figure 5-7   Labels as the combination of classification and compartments

These are used for the sensitivity of the data hold by the object(i.e. File, Directory, Network Packet) and the sensitivity clearance of the subject (i.e. processes). The information flow is decided by the sensitivity of the data and the clearance of the process.

Relationship between Sensitivity Labels

There could be three kinds of relationship between labels. Table 5-1 on page 285 shows the relationships between two Sensitivity Labels SL1 and SL2 based on the relationships between their classification and compartment.

Table 5-1   Sensitivity Label relationships

Relationship	Classification relationship	Compartment relationship
Dominance	SL1 >= SL2	SL1 compartments include all SL2 compartments.
Equal	SL1 = SL2	SL1 compartments are same as the SL2 compartments.
Disjoint	N/A	If no compartments match.

Figure 5-8 shows the relationships of the Sensitivity Labels.
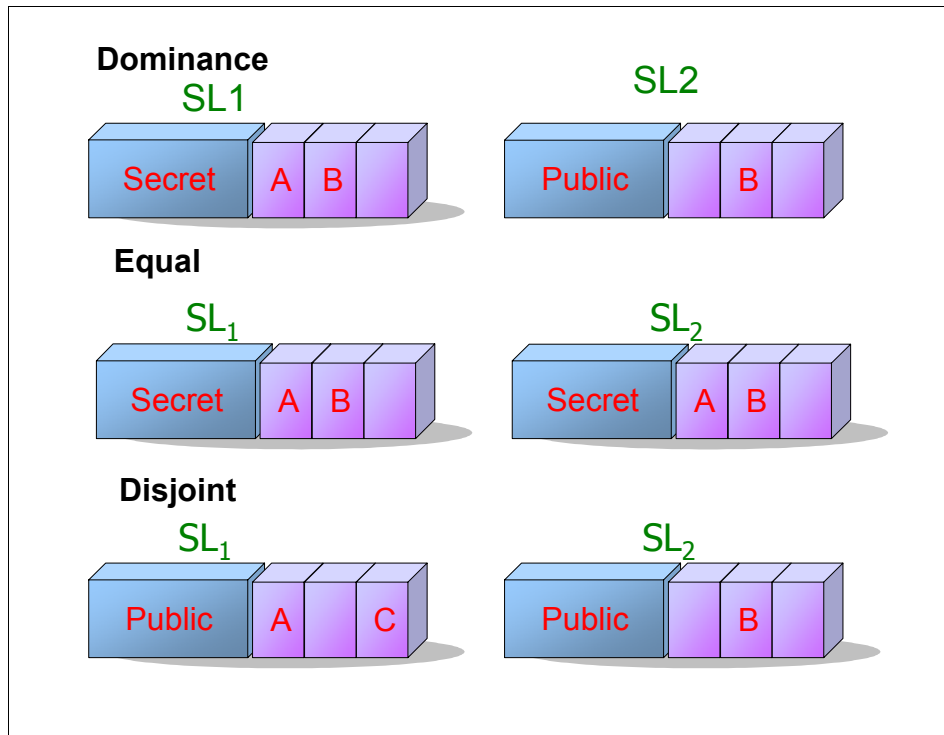
Figure 5-8   Relationship of the Sensitivity Labels

MAC enforcement rules

1. To *read* a process, SL should dominate file SL.
2. To *write* a process, SL must be equal to file SL.
3. To *execute* a process, SL should dominate file SL (same as read)
4. If SLs are disjoint, no access is allowed.

MAC security policy

► A file owner cannot change the MAC settings of the file, unless specifically authorized.

► A file owner cannot grant access to another user, unless the other user is authorized to access that class of data.

► A process cannot alter its own security attribute.

► Access to the object is determined by:

    – Sensitivity of the object (file being acted on)

    – Clearance of the subject (Acting process)

Setting the labels for the process, user, file, directory, and specifying the labels in the system is discussed in 5.5, "Configuring and managing the Trusted AIX system" on page 310.

MAC on open file descriptor

For read/write and simple file access, MAC checks are performed when a process accesses a file. Once a process has a file descriptor for the file, it can read and write the file even if the process Sensitivity Label changes to a level lower than the Sensitivity Label of the file. However, some operations, such as setting the file owner, permissions, labels, and privileges, perform access checks even after a process has obtained a file descriptor.

This means that MAC checks and partitioned directory path resolutions are not performed when a process accesses a file using a file descriptor. The Sensitivity Label of the file or process may change and access is still permitted.

5.2.3 Mandatory Integrity Control

As Trusted AIX provides access control based on the sensitivity of the information, it also provides access control based on the integrity of the object and subject. Integrity is related to unauthorized changes to the information. Integrity is the measure of the trustworthiness of the object and the subject. If the information is not trustworthy, the process should not be allowed to read that information, and in the same way if the process is not trustworthy, it should not be allowed to modify any information in the system.

Trusted AIX represents the integrity of the object or the subject using the *Integrity Labels* (TL).

What is an Integrity Label

Integrity Labels represent the level of trust in a system object or process. Integrity labels consists of only classifications (See "Classification" on page 282). Integrity labels are used for access check based on the integrity of the information held by the objects and the trustworthiness of the process accessing the information or writing the information.

Table 5-2 shows the relationships beween Integrity Labels

> **Attention:** We use the abbreviation *TL* for *Integrity Label* in many places in this chapter.

Table 5-2   Relationship between Integrity Labels

Relationship	Classification relationship
Dominance	TL1 >= TL2
Equal	TL1 = TL2
Disjoint	N/A

MIC enforcement rules

MIC is enforced whenever MAC is enforced. Additionally, MIC is enforced when a file or directory is deleted or renamed.

NOTL

There is a special Integrity Label, NOTL, that can be put on a file or process. When an object or process has an NOTL TL, no MIC checks are performed on the object or process. Only privileged users can set a TL to NOTL or change a TL if the TL is currently NOTL.

5.2.4 Other attributes

Other attributes are described as follows.

Run modes

In order to configure and maintain the system and for day-to-day operations, there are two run modes:

1. Configuration mode
2. Operational mode

When the system boots up, it initially runs in configuration mode. After initialization is complete, the run mode is changed to operational.

Configuration mode

Configuration mode is used to maintain and recover the system. Configuration mode is intended to be used for administration of critical, security-relevant parts of the system. The restrictions associated with configuration mode cannot be overridden or bypassed by any mechanism.

Operational mode

Operational mode is used for day-to-day operations.Operational mode is intended to be the standard operating mode of the system.The restrictions associated with operational mode cannot be overridden or bypassed by any mechanism.

These restrictions are:

1. The kernel security flags cannot be modified.

2. TCB objects cannot be created, modified, or deleted.

3. The Trusted Library Path cannot be modified. (See the description of Trusted Library Path in 4.4, "Trusted Execution Path, Trusted Library Path, Trusted Shell, and Secure Attention Key" on page 267.)

The only difference between operational mode and configuration mode is that in the latter, some daemons and services are not started and not all file systems may be mounted. Security mechanisms, however, such as Sensitivity Labels, Discretionary Access Controls, Mandatory Access Controls, privilege checks, and authorizations are in full force, as dictated by the relevant security configuration flags.

> **Attention:** We recommend that the system be operated in only operational mode to ensure that all expected system functionality is available.

System run mode can be displayed by the **getrunmode** command and can be set using the **setrunmode** command by a authorized user, as shown in Example 5-1.

Example 5-1   System run mode

```
$ getrunmode
System is currently in CONFIGURATION MODE.
$ setrunmode -o
System runtime mode is now OPERATIONAL MODE.
```

File Security Flags

File Security Flags are flags that affect the way the files are accessed. Trusted AIX provides the File Security Flags shown in Table 5-3.

Table 5-3   File Security Flags

Flag	Description
FSF_APPEND	The file can only be appended to and not altered in operational mode.
FSF_AUDIT	The file is marked as a part of the audit subsystem. To read/write these files, the process is required to have PV_AU_READ/PV_AU_WRITE privileges respectively.
FSF_MAC_EXMPT	Effective Privilege Set with PV_MAC_OVRRD ignores MAC restrictions when attempting to access the object.
FSF_PDIR	The directory is a partitioned directory.
FSF_PSDIR	The directory is a partitioned subdirectory.
FSF_PSSDIR	The directory is a partitioned sub-subdirectory.
FSF_TLIB	The object is marked as part of the Trusted Library. To change this property requires: 1. The system to be running in configuration mode. OR 2. The kernel security flag trustedlib_enabled is OFF.
FSF_TLIB_PROC	Processes marked as TLIB processes can link only to *.so libraries that have the TLIB flag set. To change this property requires: 1. The system to be running in configuration mode. OR 2. The kernel security flag trustedlib_enabled is OFF.

The File Security Flag can be displayed using the **lstxattr** command. It can be set using the **settxattr** command. A partitioned directory is represented by the FSF flag FSF_PSDIR in Example 5-2.

Example 5-2   displaying file security flags

```
$ pdmkdir pdir
$ lstxattr -f -F pdir
pdir:
        maxsl=IMPLEMENTATION LOW
        minsl=IMPLEMENTATION LOW
        tl=TL IMPLEMENTATION LOW
        secflags=FSF_PSDIR
```

Kernel Security Flag

Some kernel flags have been introduced to control the security policies. Using these kernel flags, various security checks can be enabled or disabled. For example, if the Sensitivity Label check has been disabled by the administrator of the system, the system will not check for the labels for determining the access permissions.

These flags are only supported when the Trusted AIX is enabled. In the user space, these flags are stored in the ODM database. Trusted AIX has different configuration of flags for different modes. Depending upon the run mode, the kernel checks for the corresponding Kernel Security Flags.

Trusted AIX provides Kernel Security Flags shown in Table 5-4.

Table 5-4   Kernel security flags

Kernel Security Flag	Enabled	Disabled	Operational mode default	Configuration mode default
tnet_enabled.	Trusted network functionality available.	Trusted network functionality cannot be configured or used.	Disabled.	Disabled.
tl_write_ enforced.	MIC enforced on write, delete and rename operations.	Configuration set so that TLs are not used for write checks.	Enabled.	Enabled.
tl_read_ enforced.	MIC enforced on read operations.	Configuration set so that TLs are not used for read checks.	Disabled.	Enabled.
sl_enforced.	MAC enforced.	Configuration set so that SLs are not used for access control.	Enabled.	Enabled.
trustedlib_ enabled.	FSF_TLIB flag on file system objects is honored.	FSF_TLIB flags are not honored.	Disabled.	Disabled.

Kernel Security Flag	Enabled	Disabled	Operational mode default	Configuration mode default
root.	Root will be enabled.	Root will be disabled.	Disabled.	Enabled.

The current flag settings can be seen by the **getsecconf** command. The command displays the setting for the mode the system is currently running in, as shown in Example 5-3.

Example 5-3   Displaying run modes

```
$ getrunmode
System is currently in CONFIGURATION MODE.

$ getsecconf
CONFIGURATION MODE Security Flags
TRUSTED NETWORK        :   DISABLED
SL ENFORCEMENT         :    ENABLED
TL WRITE ENFORCEMENT   :    ENABLED
TL READ ENFORCEMENT    :    ENABLED
TLIB                   :   DISABLED
ROOT                   :   DISABLED

$ setrunmode -o
System runtime mode is now OPERATIONAL MODE.

$ getsecconf
OPERATIONAL MODE Security Flags
TRUSTED NETWORK        :   DISABLED
SL ENFORCEMENT         :    ENABLED
TL WRITE ENFORCEMENT   :    ENABLED
TL READ ENFORCEMENT    :   DISABLED
TLIB                   :   DISABLED
ROOT                   :   DISABLED
```

Privilege

Privileges are associated with the process and can be used to bypass specific restrictions and limitations of the system. In some cases we need some applications to bypass some restrictions to make it function properly on a Trusted AIX system. We can accomplish this task by providing proper privileges to the application. Assignment of the privilege to the application for bypassing restrictions and limitations can only be done by the authorized users. Trusted AIX provides some tool to determine what all privileges are needed. The privilege assignment is based on the *least privileged principle*, that is, any process should be given only the least privileges needed to work properly. A detailed description of this subject can be found in Chapter 3, "Role Based Access Control" on page 165.

Authorization

Users are assigned some roles and those roles consist of some authorizations. An administrator can assign different roles to the users. The functionality available to the user depends on the roles provided to the user. A detailed description of roles and authorizations can be found in Chapter 3, "Role Based Access Control" on page 165.

Roles

Roles are a collection of authorizations. A role can be build up of any number of authorizations. A user can be assigned different roles that decide what functions that user is able to do in the system.

5.2.5 Introduction to Trusted Networks

The extended security attributes of enhanced security systems necessitate certain functionality for secure networking. Trusted Networking is the set of functions that process incoming/outgoing network traffic on Trusted AIX. It supports a number of recognized networking standards, including U.S. DoD RFC1108 and the Common/Commercial IP Security Option (CIPSO).

Trusted Network does two things:

1. Assigns labels to the traffic.

    a. Incoming: Assigns a label if it is not in the packet.

    b. Outgoing: Inserts the label in the packet if specified to do so.

2. Filtering traffic.

    a. Determine what is allowed in and out.

Figure 5-9 shows how incoming traffic is processed.
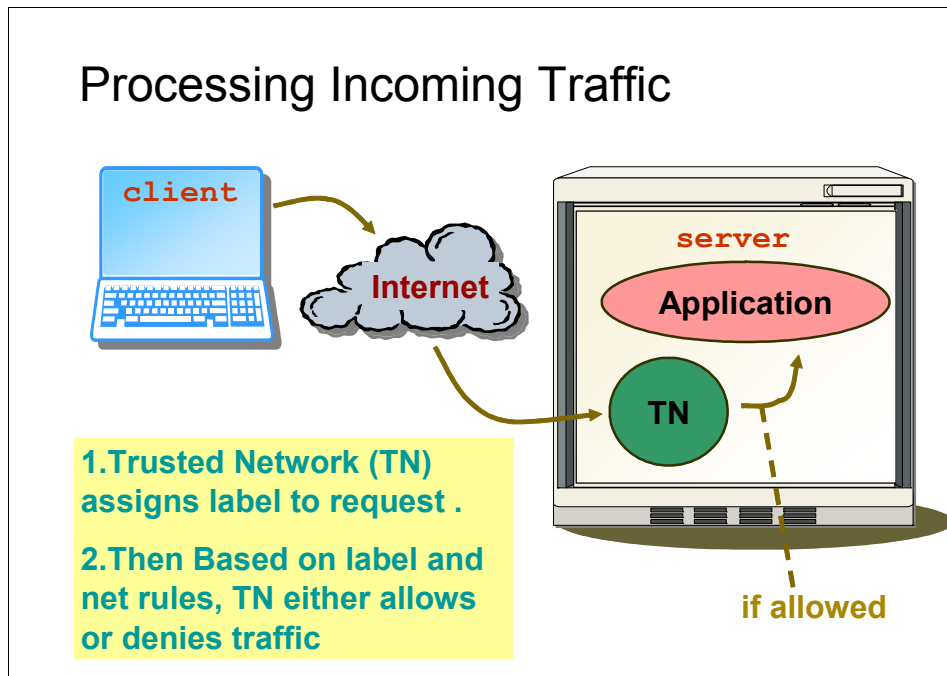
Figure 5-9   Processing incoming traffic

Figure 5-10 shows how outgoing traffic is processed.

Processing Outgoing Traffic

client

Internet

server

Application

TN

1. Server sends traffic with label. Trusted Network (TN) intercedes.

2. Based on net rules, TN allows or denies traffic

3.If specified, TN puts label on traffic

if allowed
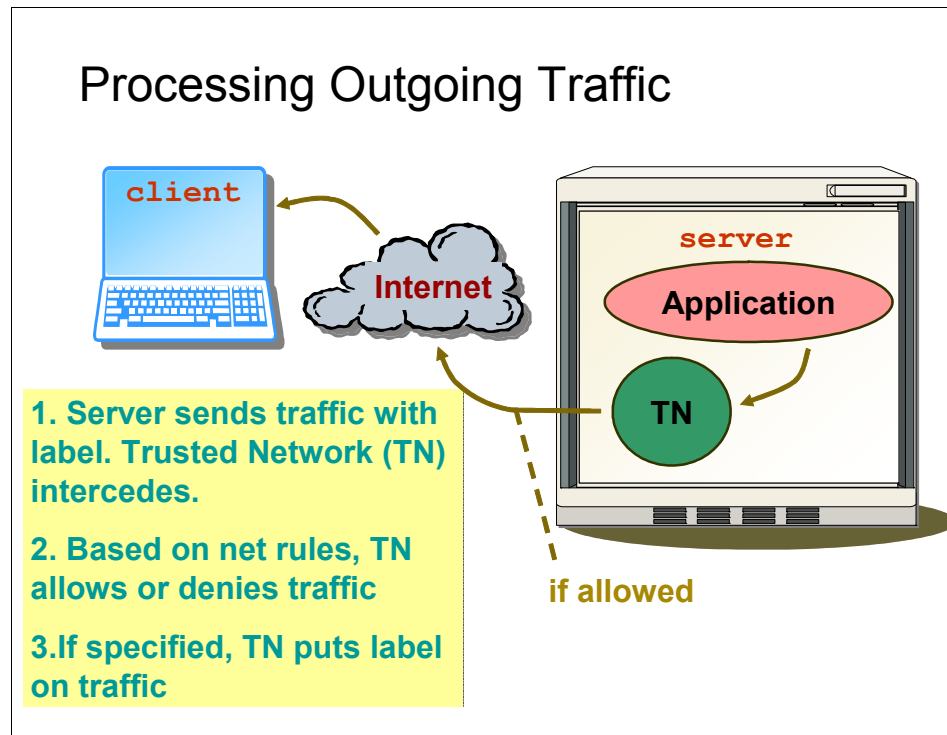
Figure 5-10   Processing outgoing traffic

Packet labels
Label information is attached to the packets in the IP header. These labels are attached to the packet in the IP layer and transmitted to the next layer. These labels depend on two protocols: RFC 1108 and CIPSO.

Network rules
There are two kinds of rules in the system. Based on these rules, a decision is made to allow or deny the packet:

1. Host rule

2. Interface rule

Host and Network Interface rules govern the way that the system deals with packets of incoming and outgoing data over the network. Host rules apply to specific hosts. Network Interface rules apply to the interfaces through which hosts connect to the network. If there are any conflicts between a host rule and an interface rule, the host rule takes precedence.

In general, the rules pertain to protocols used, ranges of addresses (both hosts and ports) to which to apply the rules, and what Sensitivity Labels to assign to the packets.

Unlabeled network traffic

This is used for most internet traffic:

► Incoming: Trusted Network assigns labels to the incoming packets and based on the labels it allows or denies the traffic.

► Outgoing: Trusted Network does not attach a label to the outgoing packet.

Labeled network traffic

This is mostly used for communication among Trusted Systems. It is required for the receiving and sending system to interpret labels.

► Incoming: The Trusted Network assigns a label to the packet based on the rules setting and decides to deny or allow the traffic.

► Outgoing: The Trusted Network inserts the labels to the outgoing packet.

Commands

Trusted AIX provides two commands for enabling and maintaining the Trusted Network:

`tninit`   This command is used to initialize the Trusted Network subsystem and maintain the rules database. The `tninit` command supports four operations: init, load, save, and display. The user must have the aix.mls.network.init authorization to run this command.

`netrule`   A command to add, remove, list, or query rules, flags, and security labels for interfaces and hosts. The user must have the aix.mls.network.config authorization to run this command.

Some examples of how these two commands can be used are shown in 5.5.6, "Trusted Network configuration" on page 317.

5.2.6 Audit subsystem

As the new security attributes have been added to the system, the audit system has been modified to audit and display the new security attributes.

The following changes have been done to the audit system:

1. Commands have been modified for selecting and displaying the MLS attributes in MLS audit records.

2. Sensitivity Label and Integrity Label (TL) ranges are specified in the /etc/security/audit/config file. All the objects between these range will be audited. The administrator can download the audit ranges.

3. The audit range is defined under the WAR stanza. One example of this is given in Example 5-4.

Example 5-4   /etc/security/audit/config file with MLS attributes

```
start:
        binmode = on
        streammode = off

bin:
        freespace = 65536
        trail = /audit/trail
        bin1 = /audit/bin1
        bin2 = /audit/bin2
        binsize = 10240
        cmds = /etc/security/audit/bincmds

stream:
        cmds = /etc/security/audit/streamcmds

classes:
        general = MLS_getWAR, MLS_setWAR, MLS_SetSyslab, MLS_SetPLab
        files = File_Write

users:
        root = files
         isso = Genral

WAR:
        obj_min_sl = SLSL
        obj_max_sl = SHSL
        obj_min_tl = SLTL
        obj_max_tl = SHTL
```

```
              sub_min_sl = SLSL
              sub_max_sl = SHSL
              sub_min_tl = NOTL
              sub_max_tl = NOTL
```

4. The `audit start` command has been changed to download the audit range on an MLS system; if it is a non-MLS system, it will not download the audit range.

5. The `auditpr` command has been modified to print the MLS attributes. There are three MLS fields in the audit header of a MLS audit record. Three new sub parameters have been added under the `auditpr -h` command to display those attributes:

   – E: Effective privilege

   – S: Effective Sensitivity Label

   – I: Effective Integrity Label (TL)

6. A new parameter, -s, has been added to the `auditpr` command to display the MLS data in the tail portion of a MLS audit record. If -s is omitted, no MLS data in the MLS audit tail is displayed. The -v parameter to `auditpr` command is unchanged. It does not display the MLS data.

7. MLS related field names for the `auditselect` command have been added, so that user can select audit records based upon the MLS fields in the headers of MLS audit records. The new MLS selection criteria are: effective privilege, effective Sensitivity Label, and effective Integrity Label (TL). These new field names will be used in the expression of the -e parameter of the `auditselect` command (see Table 5-5).

Table 5-5   New keywords

New keyword	Selection criteria
priv	Privilege in string format
tl	TL in string format
sl	SL in string format

8. The `audirstream` command has been modified to handle the MLS buffer length in the /tail audit header.

9. MLS related events have been added to the /etc/security/audit/events. Those events can be added to /etc/security/audit/config for the auditing.

Example 5-5 is an example of the /etc/security/audit/events file with the MLS events.

Example 5-5   /etc/scurity/audit/events file updated with MLS events

```
*        sec_getauditrange()
         MLS_GetWAR = printf "cid=%d"

*        sec_setauditrange()
         MLS_SetWAR = printf "cid=%d"

*        sec_setrunmode()
         MLS_SetRunMode = printf "rc=%d"

*        sec_getsecconf()
         MLS_GetSecconf = printf "conf_flag=%d, oper_flag=%d, rc=%d"

*        sec_setsecconf()
         MLS_SetSecconf = printf "rc=%d"

*        sec_settpmode()
         MLS_SetTPMode = printf "mode=%d, rc=%d"

*        sec_setsyslab()
         MLS_SetSyslab = printf "err=%d"

*        sec_getpsec()
         MLS_GetPsec = printf "pid=%d, err=%d"

*        sec_setplab()
         MLS_SetPLab = printf "pid=%d, err=%d"

*        sec_setptlibmode()
         MLS_SetPTLIBMode = printf "mode=%d, err=%d"
```

Now we add the config file above to our system and try to audit the MLS_SetPLab event. We will use the `auditpr` command to list the MLS attributes with the new options E, S, and I, which are introduced for MLS attributes.

Only a user having the aix.ras.audit authorization can start, stop, or query the audit subsystem. Only that user can print the report on the stdout (see Example 5-6).

Example 5-6   Audit subsystem authorization

```
$ audit start
$ settxattr -p effsl=SEC $$
$ audit shutdown
auditing reset
$auditpr -i /audit/trail -h E,S,I
Privilege            SL              TL
------------------------------------------------
PV_ROOT              SEC             TL_IMPL_LO
```

5.2.7 Partitioned directory

In Trusted AIX, the directories have a range of Sensitivity Labels. Any process having an SL between that range can list the files inside that directory. Although a user at a lower SL cannot access the files at a higher SL, being able to list the file names could still gain information about them. To stop this kind of information leak, a new feature has been added in the Trusted AIX called *partitioned directory*. A partitioned directory has hidden subdirectories corresponding to each SL.

There are two partitioned directory access modes in the system:

1. Virtual mode

2. Real mode

In virtual mode, the user will not be able to see the hidden subdirectories. In the real mode, all the directories will behave same as the regular directory structure and the hidden directories will be visible to the user. By default, the system is in virtual mode. Mode change can only be done by an aix.mls.pdir.mode authorized user.

There will be one hidden subdirectories corresponding to each Sensitivity Label. Whenever a user used the **cd** command to change to a partitioned directory, he will be redirected to the subdirectory corresponding to his or her or her SL, so he will only be able to see the files inside the hidden subdirectory corresponding to his or her process effective SL. The redirection is not apparent to the user. Users will be allowed to see only those files whose labels match that of the process labels.

Partitioned directories can be created by the **pdmkdir** command and can be removed using the **pdrmdir** command. Partitioned directory access mode can be changed by the **pdmode** command. For linking a file in a hidden directory to other hidden directories that are at a higher SL, the **pdlink** command is used. A regular directory can be converted to the partitioned directory using the **pdset** command.

You can make the pdir using the **pdmkdir** command:

```
$ pdmkdir pdir
```

If we create the file inside the partitioned directory, it will be created under a hidden directory:

```
$ touch pdir/file
```

We can change the Sensitivity Label of the process and create the file with same name again because it will be created inside another hidden directory based on the effective SL of the process:

```
$ settxattr -p effsl=SECRET $$
$ touch pdir/file
$ ls pdir
file
```

If we list the files inside the partitioned directory in the real mode using the **pdmode** command, we will see the hidden directories, because in read mode redirection is not done:

```
$ pdmode -r ls pdir
000000  07P000
```

Partitioned directories can be very useful in the environment where we do not want the person operating in the lower level to see the files owned by others that are at a different level.

A person can share his or her file to other person at the higher level by linking its file using the **pdlink** command. The **pdlink** command creates a link to the files to all the hidden directories that are of a higher level and exist at the time of creation.

5.3 Applications on Trusted AIX

There are two types of applications on Trusted AIX:

► Trusted application
► Untrusted application

Trusted applications are those that require privileges to run, that is, trusted applications specifically check privileges and authorization, and grant access to only those users with the required access controls in place.

Untrusted applications are those that do not require privileges to run on the system. Virtually all commercial off-the-shelf applications that run on Trusted AIX are untrusted applications. Even with untrusted applications, Trusted AIX applies default labels and checks to make sure that untrusted applications cannot bypass the system security mechanism.

Setup for porting applications on Trusted AIX
Unfortunately, not all commercial off-the-shelf applications are "well behaved", that is, they perform various functions that violate the system security policy of the operating system. Some examples are applications that must execute as superuser or root account. Trusted AIX has provided tools and features to run such programs, while still minimizing the threats to system security:

► A fine-grained set of privileges that can be used by the isso authorized user to assign privileges to an application on a case-by-case basis so that the application functions correctly on the system in a minimal-risk, secure manner. The principle of least privilege requires that application programs be assigned only the minimal set of privileges necessary to allow the application program's performance.

► A superuser emulation feature to allow those applications that assume the superuser status to execute within the control of the Mandatory Access Control (MAC) and Mandatory Integrity Control (MIC).

► A privilege-finding command, `tracepriv`, that shows the privileges used by a given process.

Determining and assigning privileges
Many applications that run on your system will require privileges to run.You can grant these privileges most easily by placing them in the executable binary's innate privilege set. Thus, when the binary is executed, the resulting process inherits the privileges it needs to run.

You need to set the minimum necessary privileges on the file to keep it as secure as possible. Any privileges granted beyond those necessary for proper execution violates the least privileged principle. Therefore, an important step in mounting an application on your system is to determine the minimum required privileges.

To find the privilege needed for an application as an isso authorized user, assign PV_ROOT to your shell and exercise the application vigorously and see what privileges it has used. After you have executed and exercised an application, and before you exit the application, run the `tracepriv` command on the application's process. The system returns a list of all privileges used by the application since it started. Add the command to the privilege command database with these privileges in the innate privilege set. After you have done this, grant yourself no privileges and run the application again, to make sure that it still runs properly.

In Example 5-7, for the script file `myscript.sh`, we show how to trace all privileges needed to successfully run it.

Example 5-7   Tracing privileges

```
$ setsecattr -p eprivs=PV_ROOT mprivs=PV_ROOT $$
$ cat myscript.sh
mkdir testdir
$ tracepriv ./myscript.sh

294968: Used privileges for /usr/bin/sh:
  PV_AZ_ROOT                        PV_DAC_R
  PV_PROC_PRIV                      PV_TP_SET
  PV_KER_ACCT                       PV_KER_RAC

$ setsecattr -c
innateprivs=PV_AZ_ROOT,PV_DAC_R,PV_TP_SET,PV_PROC_PRIV,PV_KER_ACCT,PV_K
ER_RAC $PWD/myscript.sh
```

5.4 Installation of Trusted AIX

Trusted AIX can only be enabled when you initially install the AIX operating system. The option appears as part of the operating system installation process. Currently, only the preservation, New, and Overwrite installation options are supported for Trusted AIX. Migration installation to Trusted AIX is not supported. For the preservation installation, the file system should be JFS2 file system.

We now show the preservation installation from AIX 5L V5.3 to Trusted AIX and a New and Complete Overwrite installation of the Trusted AIX.

Preservation installation from AIX 5L V5.3 to Trusted AIX

To check whether the preservation installation from AIX 5L V5.3 to Trusted AIX works properly, we will create some user IDs on the AIX 5L V5.3 system and create some data for those users (user1 and user2) to check that the data is preserved. User information for these two users is shown in Example 5-8. We will create some other non-system file systems and one volume group other than rootvg named testvg to check that the file systems and volume groups are preserved. We will create some logical volumes and file systems in rootvg as well as in testvg and check whether they will be preserved in the preservation installation.

Example 5-8   User ID information

```
user1:
        admin = false
        login = true
        rlogin = false
        su = false

user2:
        admin = true
        login = true
        rlogin = true
        su = true
```

We have four file systems with the configuration shown in Example 5-9.

Example 5-9   File systems information

```
MountPoint:Device:Vfs:Nodename:Type:Size:Options:AutoMount:Acct
/home/user1/fs1:/dev/fslv00:jfs2:::32768:rw:no:no
/home/fs2:/dev/lv00:jfs:::32768:rw:no:no
/fs3:/dev/lv01:jfs:::32768:rw:no:no
/fs4:/dev/fslv01:jfs2:::32768:rw:no:no
```

We have two volume groups with the information shown in Example 5-10.

Example 5-10   Volume group information

```
testvg:
LV NAME TYPE       LPs   PPs   PVs  LV STATE      MOUNT POINT
loglv00 jfs2log   1     1     1    open/syncd    N/A
fslv00 jfs2       2     2     1    open/syncd    /home/user1/fs1
rootvg:
LV NAME TYPE       LPs   PPs   PVs  LV STATE      MOUNT POINT
lv00              jfs   2     2     1    open/syncd    /home/fs2
lv01              jfs   2     2     1    closed/syncd  /fs3
fslv01            jfs2  2     2     1    closed/syncd  /fs4
```

The user is storing data in /home, as shown in Example 5-11.

Example 5-11   User data

```
# cd /home
# ls
esaadmin    guest        lost+found   user1        user2
# cd /home/user1
# mkdir dir1 dir2 dir3
# touch file1 file2 file3
# ls
.profile  dir1      dir2      dir3      file1     file2     file3
fs1
# cd /home/user2
# mkdir dir1 dir2 dir3
# touch file1 file2 file3
# ls
.profile  dir1      dir2      dir3      file1     file2     file3
```

Now we try to do a preservation installation to see whether this information is preserved. We see how the additional attributes are set.

Follow these steps to install Trusted AIX from CD or through a NIM installation:

1. Select the system console by pressing F1 (or 1 on a ASCII terminal) and press Enter.

2. Select the English language for the base operating system (BOS) installation menus by typing 1 in the choice field. Press Enter to open the Welcome to Base Operating System Installation and Maintenance screen.

3. Type 2 to select 2 Change/show Installation setting and Install in the choice field and press Enter (see Figure 5-11).

```
                    Welcome to Base Operating System
                    Installation and Maintenance

Type the number of your choice and press Enter. Choice is indicated by >
>>.

>>> 1 Start Install Now with Default Settings

    2 Change/Show Installation Settings and Install

    3 Start Maintenance Mode for System Recovery

    4 Configure Network Disks (iSCSI)




    88  Help ?
    99  Previous Menu

>>> Choice [1]: 2
```

Figure 5-11   Welcome menu for Base Operating System Installation and Maintenance

4. In the Installation and Settings screen, verify that the installation settings are correct by checking the method of installation (see Figure 5-12).

```
                          Installation and Settings

Either type 0 and press Enter to install with current settings, or type
the
number of the setting you want to change and press Enter.

    1  System Settings:
           Method of Installation.............Preservation
           Disk Where You Want to Install.....hdisk0...

    2  Primary Language Environment Settings (AFTER Install):
           Cultural Convention................C (POSIX)
           Language...........................C (POSIX)
           Keyboard.........,.................C (POSIX)

    3  Security Model......................Default
    4  More Options  (Software install options)

>>> 0  Install with the settings listed above.

                          +-----------------------------------------------
-----
    88  Help ?          |  WARNING: Base Operating System Installation wi
ll
    99  Previous Menu  |destroy or impair recovery of SOME data on the
                       |destination disk hdisk0.
>>> Choice [0]: 3
```

Figure 5-12   Installation and Settings screen

Restriction: For preservation installation, the file system should be JFS2. If it is JFS, it will fail because JFS does not provide Extended Attributes V2 for security labels. We cannot have migration installation for Trusted AIX because we cannot change the Security Model in a migration installation.

5. To change the System settings, type 1 in the choice field and hit Enter. Change the method of installation to preservation.

6. To install Trusted AIX, enter 3 in the choice field and hit Enter. By default, the Trusted AIX option is No. Change it to Yes by entering 1 in the choice field of the Security Models screen (see Figure 5-13).

```
                        Security Models

  Type the number of your choice and press Enter or
  press Enter to continue to next choices.

  1. Trusted AIX.......................................... no

  2. Other Security Options  (Trusted AIX and Standard)
       Security options vary based on choices.
       LSPP, SbD, CAP/CCEVAL, TCB



>>> 0  Continue to more software options.

     88  Help ?
     99  Previous Menu

>>> Choice [0]: 1
```

Figure 5-13   Security Model Screen

7. To enable the configuration for certification, select 2 and then select 1.

8. Select 0 to install any other software; when finished, come back to the Installation and Settings screen and continue with the installation.

Trusted AIX installation by default creates three different users with three different roles. The detailed description can be found in Chapter 3, "Role Based Access Control" on page 165. These three users are assigned three different roles:

► ISSO: Information System Security Officer

► SA: System Administrator

► SO: System Operator

After the installation is complete, you will be prompted to choose the terminal type. After choosing the terminal type, the Installation Assistant screen will appear, where you can set the password for these three users created by default and do some other configurations.

Log in to the system as an isso user and run `swrole` to change to the isso role. Now we check for the data we had in the system before installation.

Both the volume groups are preserved (see Example 5-12).

Example 5-12   Listing the volume group information

```
$ lsvg -l testvg
testvg:
LV NAME            TYPE      LPs   PPs   PVs  LV STATE      MOUNT
POINT
loglv00            jfs2log   1     1     1    closed/syncd  N/A
fslv00             jfs2      1     1     1    closed/syncd
/home/user1/fs1

$ lsvg -l rootvg
rootvg:
lv00               jfs       1     1     1    closed/syncd  /home/fs2
lv01               jfs       1     1     1    closed/syncd  /fs3
fslv01             jfs2      1     1     1    closed/syncd  /fs4
```

We need to mount the file system (Example 5-13).

Example 5-13   Mounting the file system after the preservation installation

```
$ mount /home/user1/fs1
Replaying log for /dev/fslv00.
$ mount /home/fs2
Replaying log for /dev/lv00.
$ mount /fs3
mount: /dev/lv01 on /fs3: No such file or directory
$ mount /fs4
mount: /dev/fslv01 on /fs4: No such file or directory
```

Because the mount point for two file systems were in the root file system and the root file system is not preserved, we need to create the mount point and mount those file systems again (Example 5-14).

Example 5-14   Creating the mount point and mounting the file systems

```
$ mkdir /fs3
$ mount /dev/lv01 /fs3
$ mkdir /fs4
$ mount /dev/fslv01 /fs4
```

User definitions are not preserved (Example 5-15).

Example 5-15   User definition is not preserved

```
$ lsuser user1
3004-687 User "user1" does not exist.
$ lsuser user2
3004-687 User "user2" does not exist.
```

We need to create the user and assign them the Sensitivity Label and Integrity Label (see "User account creation" on page 330 for more information).

By default, the users are assigned System Low Sensitivity Label (SLSL) and System Low Integrity Label (SLTL) (Example 5-16). The isso user has to customize labels for the users (see 5.5.4, "User Account configuration" on page 314 for more information).

Example 5-16   Default values assigned to the user

```
default:
         minsl = "SLSL"
        maxsl = "SLSL"
        defsl = "SLSL"
        mintl = "SLTL"
        maxtl = "SLTL"
        deftl = "SLTL"
```

User data is also preserved in the system. Because the user definition is not preserved, we need to create the user and change the ownership of his or her data as before.

> **Important:** Once Trusted AIX is enabled, it cannot be disabled. Evaluate your need for Trusted AIX before choosing this mode.

New and Complete overwrite installation

Steps for a New and Compete overwrite installation are the same as for a preservation installation. The only difference is that we have to change the Installation type. After selecting the current terminal as your console and English as the language of installation, follow the following steps.

1. Type 2 to select 2 Change/show Installation setting and Install in the choice field and press Enter.

2. In the Installation and Settings screen, verify that the installation settings are correct by checking the method of installation.

3. To change the System settings, type 1 in the choice field and press Enter. Change the method of installation to New and Complete overwrite.

4. For installing the Trusted AIX, enter 3 in the choice field and hit Enter. By default, the Trusted AIX option is No. Change it to Yes by entering 1 in the choice field of the Security Models screen.

The rest of the steps are same as described in "Preservation installation from AIX 5L V5.3 to Trusted AIX" on page 303. In a New and Complete overwrite installation, all the system as well as user data will be lost.

5.5 Configuring and managing the Trusted AIX system

Here we discuss configuring and managing the Trusted AIX system.

5.5.1 Disabling root

Root will be disabled in the Trusted AIX system. This is primarily to minimize the damage that can be caused by a single user with all privileges.

All types of system logins as root user are disabled. Only the **su** command allows root user logins. Processes owned by root are not assigned any special privileges. The root-owned setuid and non-setuid programs work as before when used by authorized users. For unauthorized users, the program will run if the DAC modebits or ACLs allow execution, but the program will not be assigned any privileges, so the program may not be able to perform privileged operations when run by unauthorized users. Therefore, it is necessary to assign proper privileges to newly installed applications if the applications need to perform privileged operations (see 5.3, "Applications on Trusted AIX" on page 301 for information).

Trusted AIX by default has three roles. The system administrative tasks are distributed between these three roles. These three roles are.

► isso: Information System Security Officer

► sa: System Administrator

► so: System Operator

The system can be administered by the users with these three roles. It is not advised to assign all three roles to one user.

5.5.2 System configuration

The Trusted AIX has two kind of run modes: operational mode and configuration mode. The system can be configured to enable or disable various security checks in the system based on the policy decision.

The security checks can be disabled or enabled only in Configuration Mode using the `setsecconf` command. A user with aix.system.config.write authorization or a user with isso role can do the configuration of the system. The following are the options for system configuration:

tnet   Trusted Network.

tlwrite   Integrity Label (TL) enforcement when a process tries to write to a file system object.

tlread   If enabled, TLs are checked on the read operation.

sl   Mandatory Access Control is enforced if this flag is enabled.

tlib   If enabled, then FSF_TLIB will be honored.

To change the system configuration, first we need to switch to the configuration mode and then run the `setsecconf` command. The setting after the command completion will be displayed.

For example, if the sl attribute has been disabled, then the Sensitivity Label will not be checked for access decisions.

Example 5-17   How to set the system configuration

```
$ setrunmode -c
System runtime mode is now CONFIGURATION MODE.

$ setsecconf -c sl=enable tlread=enable tlwrite=enable
CONFIGURATION MODE Security Flags
TRUSTED NETWORK          :   DISABLED
SL ENFORCEMENT           :    ENABLED
TL WRITE ENFORCEMENT     :    ENABLED
TL READ ENFORCEMENT      :    ENABLED
TLIB                     :    ENABLED
ROOT                     :   DISABLED
```

5.5.3 Label configuration

All the labels are stored in a Label Encoding file. All the applications use this file to convert the binary format of the label to human readable strings.

Label Encoding file

The label strings and its corresponding values are stored in a plaintext file that is called the Label Encoding file, and is stored in the /etc/security/enc/LabelEncodings path. This file follows the MITRE standard format. The definition from this file is loaded into the kernel using the initlabeldb API. Every command and API uses this file for the mapping between the human readable string and the corresponding binary value.

Labels can be customized for each site after the installation of the Trusted AIX by modifying this file by an isso user. No command line is provided for modification of the Label Encoding file directly. For modifying the LEF file, the existing LEF file is copied to a temporary file using the `labck` command. The temporary file can be edited using any editor. While editing, we need to take care of the standards the LEF file follows.

A Trusted AIX system defines a SYSTEM_LOW sensitivity and integrity label that is dominated by all the labels and a SYSTEM_HIGH sensitivity and integrity label that dominates all the labels. These definitions take the values of the highest and lowest Sensitivity Labels as defined in the Label Encoding file.

The system also defines the accreditation range in the Label Encoding file. Users cannot have labels that are not in the *Accreditation Range* section.

The System Labels can be seen by the `getsyslab` command (Example 5-18). The system labels can be changed by isso or an aix.mls.system.label.write authorized user.

Example 5-18   Getting system labels

```
$ getsyslab
System Minimum Sensitivity Label  : IMPLEMENTATION LOW
System Maximum Sensitivity Label  : TOP SECRET ALL
System Minimum Integrity Label    : TL IMPLEMENTATION LOW
System Maximum Integrity Label    : TL TOP SECRET
```

This file is protected by permission bit 000 and with SYSTEM Height SL and TL.

Format of the LabelEncodings file

The label encoding contains versions and the following mandatory sections. Each section should start with one of these section keywords followed by colon:

1. Classifications

2. Information labels

3. Sensitivity Labels

4. Clearances

5. Channels

6. Printer banners

7. Accreditation range

The Label Encoding file starts with the VERSION entry. This is a sequence of characters that may contain white spaces.

Each of the following keywords can be present in a section. These keywords terminate with a semi-colon (;):

► name=name

  Keyword to define the full name of the classification or compartment.

► sname=name

  Keyword to define an abbreviated name. Optional.

► aname=name

  Alternate keyword for the classification. Optional.

► value=value

  Keyword to specify the internal integer value of the classification or compartment.

► compartments=bit

  Keyword to specify which compartment bit must be 0 or 1 if the word is present in the label.

Example 5-19 shows the Classification section for the Labels.

Example 5-19   Version and Classification section of Label Encoding File

```
**************************************************
VERSION= AIX VERSION
**************************************************


**************************************************
CLASSIFICATIONS:
**************************************************


          name= IMPLEMENTATION LOW;      sname= IMPL_LO; value= 0;
          name= UNCLASSIFIED;            sname= U;       value= 20;
          name= PUBLIC;                  sname= PUB;     value= 40;
          name= SENSITIVE;               sname= SEN;     value= 60;
          name= RESTRICTED;              sname= RES;     value= 80;
          name= CONFIDENTIAL;            sname= CON;     value= 100;
          name= SECRET;                  sname= SEC;     value= 120;
          name= TOP SECRET;              sname= TS;      value= 140;
```

5.5.4 User Account configuration

A user with the System Administrator role can create a user, but the password, labels and other security attributes are assigned by the user with the isso role.

User labels

Every user in the system is assigned minimum and maximum clearance labels. A user can operate only between this label range. Every user is also assigned one default label. By default, that user will be operating in that default label. As the user log ins to the system, that default label is assigned to the process.

The labels can only be assigned by a user with the isso role using the `chuser` command:

```
chuser -a minsl=IMPL_LO maxsl=TS defsl=IMPL_LO bob
```

A user can only see his or her label using the `lsuser` and `lssec` commands, but cannot change his or her label. To see other user labels, a user should have aix.mls.clear.read authorization. To modify the clearance, the user should have aix.mls.clear.write authorization.

The security labels are stored in the stanza for the user in the /etc/security/user file (Example 5-20).

Example 5-20   Labels stored in /etc/security/user file under the user's stanza

```
bob:
         minsl = "IMPL_LO"
        maxsl = "TS"
        defsl = "IMPL_LO"
        mintl = "SLTL"
        maxtl = "SHTL"
        deftl = "SLTL"
```

User login

To log in, all of the following dominance rules must be true:

► The minsl value must be dominated by the defsl value.

► The defsl value must be dominated by the maxsl value.

► The mintl value must be dominated by the deftl value.

► The deftl value must be dominated by the maxtl value.

You can specify the desired effective sensitivity and integrity labels at the time of login using the –e and –t options of the `login` command.

To log in at a sensitivity level that is not in the accreditation range of the system, you must have the aix.mls.outsideaccred authorization.

Trusted AIX does not allow system users (users with a uid less than 128) to log in.

If a user uses the `su` command to switch to some other user label, the invoker should dominate the new user.

Process labels

When the user log ins to the system, the user labels are assigned to the process. The user's default Sensitivity Label will be assigned as the effective Sensitivity Label of the process.

When a process forks, the child will inherit the labels from the parent process. Process labels can be changed by the `settxattr` command with the -p flag:

```
settxattr -p effsl=IMPL_LO efftl=TL_IMPL_LO $$
```

File and directory labels

All the objects in the system are labeled as the system starts. When a user creates file system objects, the labels of the objects will be set to that of the process effective Sensitivity Label. The labels of the files and directories can be set using the **settxattr** command:

```
settxattr -f sl=IMPL_LO filename
```

Directories have a range of Sensitivity Labels assigned, that is, minsl and maxsl, but has only one Integrity label:

```
settxattr -f minsl=IMPL_LO maxsl=SEC dirname
```

A process can access the file in the directory only if the process effective Sensitivity Label dominated both the minsl and maxsl of the directory.

IPC and devices

All the IPC objects also have labels assigned to them. Devices have a range of Sensitivity Labels. The labels can be listed and set using the **lstxattr** and **settxattr** commands.

5.5.5 Terminal configuration

Every terminal in Trusted AIX is assigned one range of labels. A user can log in to that terminal if his or her default Sensitivity Label is within the range of the terminal Sensitivity Label and is within the system accreditation range. After login, the user's labels are assigned to the terminal.

The labels for the terminal are stored in the /etc/security/login.cfg file. By default, all the terminals are assigned a range of SYSTEM_LOW SL and SYSTEM_HIGH SL and Integrity Label (TL) value of NOTL, which means every one can log in to the system having a valid Sensitivity Label.

The administrator of the system should configure the Sensitivity Labels and Integrity Labels according to the site policy (Example 5-21).

Example 5-21   /etc/security/login.cfg for terminals

```
default:
        sak_enabled = false
        logintimes =
        logindisable = 0
        logininterval = 0
        loginreenable = 0
        logindelay = 0
        minsl = "SLSL"
        maxsl = "SHSL"
        tl = "NOTL"
```

5.5.6 Trusted Network configuration

A Trusted Network can be configured using the `netrule` and `tninit` commands. The network can be configured based on the interface and based on the host. There could be separate rules for different interfaces and hosts.

The `netrule` command is used for configuring the interface rules and host rules. Using `netrule`, we can add, delete, query, or list host or interface rules. A network rule contains the following information:

► Name of the interface or the host.

► The range of the labels of the incoming packet that will be accepted.

► The default label that will be assigned to the incoming packet if it is not labeled.

► Whether we have to drop or allow the packet from that interface or that host.

   The drop flag can have three values:

   – r: Required to drop.

   – n: Not dropped.

   – i: Use interface rule.

► The protocol it uses for sending the security information is either RFC 1108 (RIPSO) or CIPSO. The type of protocol is represented by the received and transmit flags.

The received flag can have following values:

- r: RIPSO only.
- c: CIPSO only.
- e: Either CIPSO or RIPSO.
- n: Neither RIPSO nor CIPSO (system default).
- a: No restriction.
- i: Use interface/system default.

The transmit flag can have the following values:

- r: RIPSO placed on all outgoing packet IP headers.
- c: RIPSO placed on all outgoing packet IP headers.
- i: Use interface default (host default, host only).

► The RIPSO/CIPSO options include Classification, Protection Authority Flags (PAF)[1], and Domain of interpretation (DOI)[2] and Tags.

► RIPSO is basically for DoD and CIPSO is for commercial purposes.

► Trusted AIX supports three tag types: 1, 2, or 5.

Host rule

We may have situation where we want to accept or drop all the packets from some host in a specific port range. We may have to decide what protocol will be used for the security information and what security attributes will be added based on those protocols. If the host rule is not defined for a host, it will use the interface rule of the interface on which it will be accepting the packet.

Here are some examples of the host rules. Example 5-22 shows a rule for accepting all the CIPSO packets coming from the host 9.124.101.51 between the ports 13 to 30.

Example 5-22   Host rule for accepting a packet for a host

```
$ netrule h+i 9.124.101.51 =tcp :13 :30 127.0.0.1 -dn -fc:c +impl_lo
+ts +impl_lo
IN     SRC: 9.124.101.51/32 tcp:13-30  DST: 127.0.0.1/32 hopopt
       n:c:c:   | IMPL_LO | TS | IMPL_LO |
       DOI: 0x00001000
       tagset: 1,2,5
```

---

[1] PAF represents a different authority. This is basically for RIPSO.

[2] DOI: A group of systems use the same "Domain of interpretation" to indicate that they agree on the meaning of the particular value in the security option, such as classification and tags. This is for CIPSO.

Example 5-23 shows a rule for dropping all the packets coming to the port 23.

Example 5-23   Setting a host rule that drops all the packets coming from port 23

```
$ netrule h+i 9.124.101.51 =tcp :23 127.0.0.1 -dr +impl_lo +ts +impl_lo
IN      SRC: 9.124.101.51/32 tcp:23     DST: 127.0.0.1/32 hopopt
        r:i:i:   | IMPL_LO | TS | IMPL_LO |
```

Interface rules

We can set the network rule for the specific interface. When ever a packet arrives at the interface, it will be checked for both the host rule and the interface rule. host rule takes precedence over interface rules.

We can add and list the interface rules using the `netrule` command. The list of the interfaces can be seen by the `ifconfig -a` command. The current rules can be listed with the `netrule il` command.

Example 5-24   Listing the interface rules

```
$ netrule il
en0:    n:i:n:   | IMPL_LO | TS ALL | IMPL_LO |
lo0:    n:i:n:   | IMPL_LO | TS ALL | IMPL_LO |
```

We give one example of setting an interface rule. If we want to drop all the packets coming from the lo0 interface and want to change the sensitivity range and default label of the interface, we can do that using the `netrule` command. The default value for DOI is 0x00001000.

Example 5-25   Add a interface rule

```
$ netrule i+u lo0 -dr -DOI=4096 -tags=1 +U +SEC +U
lo0:    r:i:n:   | U | SEC | U |
```

> **Note:** If we have both a host rule and interface rule, the host rule takes precedence.

Initializing the Trusted Network System and loading, saving, and displaying the rules requires the `tninit` command. Run the following command to initialize the Trusted Network System:

```
$ tninit init
```

This loads the tables into the kernel that are responsible for making the translation between a local representation of an Sensitivity Label (SL) and what is transmitted over the network. Load, save, and display options are used to load the rules, save the rules in a file, and display the rule to the stdout. A file name should be specified with all these with the mapping.

Incoming traffic processing

Once the packet comes into the system, do the following:

1. Check the drop flag in the IP header of the packet.

2. Set SL on request.

3. Allow or deny based on the SL range. SL on request must be within the minSL and maxSL range.

Figure 5-14 shows the incoming traffic processing process.
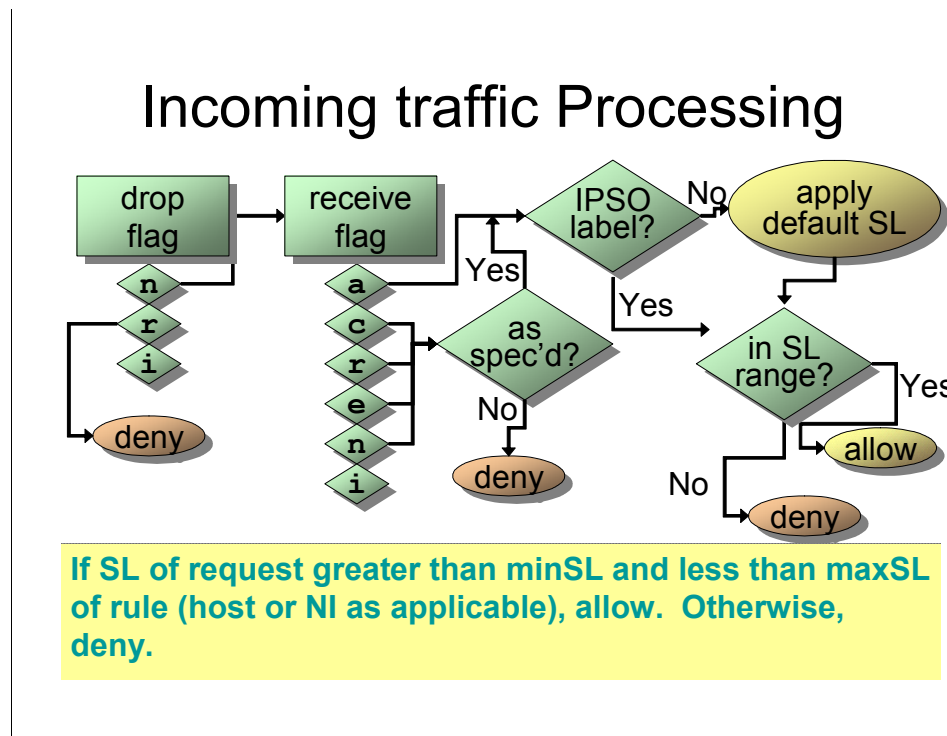
Figure 5-14   Incoming traffic processing

Outgoing traffic processing

1. Check the drop flag in the IP header of the packet.

2. Set SL on request.

3. Send or drop based on the range of the SL range. SL on request must be with in the minSL and maxSL range.

Figure 5-15 shows the outgoing traffic processing.

Outgoing traffic processing

SL of request must be within range specified in applicable host rule or NI rule.
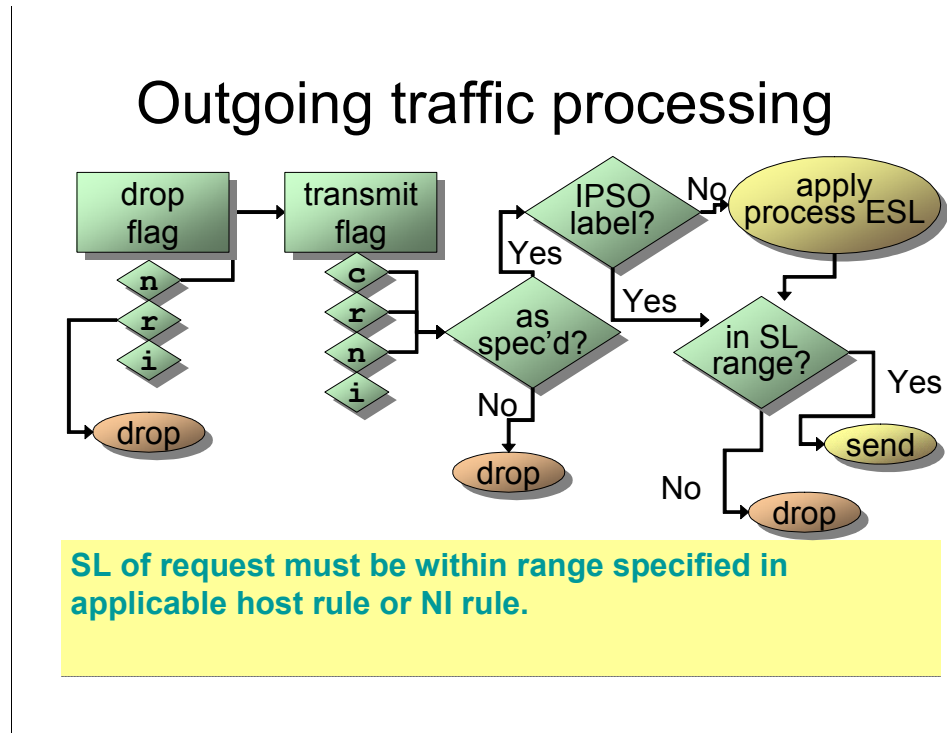
Figure 5-15   Outgoing traffic processing

Attention: The networking capabilities of AIX Trusted Network have been carefully designed to allow any conceivably desired configuration. However, you should not change the configuration from the default values in /etc/security without a thorough understanding of how AIX Trusted Networking works.

5.5.7 File system configuration

The primary file system supported by Trusted AIX is the JFS2 file system with Extended Attributes V2. Multi Level Security attributes of the objects are stored as the Extended Attributes.

A non-JFS2 file system object will not have security attributes associated with it. When the non-JFS2 file system is mounted over a JFS2 file system mount point, security attributes are used for access checks, because non-JFS2 file system objects do not have eav2 to store security attributes. While listing the security attributes for the object, the security attributes of the mount point will be displayed. The non-JFS2 file systems are called a single level file system (Example 5-26).

Example 5-26   Single level file system

```
$ crfs -v jfs -g rootvg -m /mlsfs -a size=16M
$ mount /mlsfs
$ lstxattr -f -F /mlsfs
/mlsfs:
        maxsl=SECRET
        minsl=SECRET
        tl=TL IMPLEMENTATION LOW
        secflags=
$ touch /mlsfs/file
$ lstxattr -f -F /mlsfs/file
/mlsfs/file:
        sl=SECRET
        tl=TL IMPLEMENTATION LOW
        secflags=
$ mkdir /mlsfs/dir
$ lstxattr -f -F /mlsfs/dir
/mlsfs/dir:
        maxsl=SECRET
        minsl=SECRET
        tl=TL IMPLEMENTATION LOW
        secflags=
```

When a JFS2 file system is mounted over a JFS2 file system, it will be a Multilevel File System, because the object security attributes will be used for access checks, not the mount point security attributes. When listing the security attributes the object attributes will be displayed.

Example 5-27   Multilevel File System

```
$ crfs -v jfs2 -g rootvg -m /mlsfs -a size=16M
$ lstxattr -f -F /mlsfs
/mlsfs:
        maxsl=SECRET
        minsl=SECRET
        tl=TL IMPLEMENTATION LOW
        secflags=
$ touch /mlsfs/file
$ lstxattr -f -F /mlsfs/file
/mlsfs/file:
        sl=UNCLASSIFIED
        tl=TL IMPLEMENTATION LOW
        secflags=
```

A file system can be mounted from a non-Trusted AIX system to a Trusted AIX system. The NFS mount will be Single Level Mount, that is, a security attribute of the mount point will be used for access checking.

5.5.8 Printer configuration

Once the printer has been installed on the system, the user with the isso role assigns a label range to the printer. MAC access to the printer is determined by the process Sensitivity Label and the label range assigned to the printer. The process Sensitivity Label is printed on the banner, header/footer, and trailer pages. Labels at the header/footer of each page should be dominated by the Label on the banner page. The process running the `lp` command must have MAC, MIC, and DAC access to the file it is trying to print.

Labels to the printer are assigned using the `lpadmin` command:

```
$ lpadmin -p laser -J SECRET -L IMPL_LO
```

This sets the label range of the printer named laser as IMPL_LO to SECRET, which means only processes that has effective Sensitivity Label between this range can submit print jobs to the printer laser. Any process having an effective Sensitivity Label out of this range will be denied permission to submit print jobs.

A label check is done every time a user submits the print job using the `lp` command and before the submitted job is printed by the `lpsched` daemon.

5.6 Trusted AIX scenario

All too often the cyber criminals robbing the financial service industry actually work for them. Some of the most notorious recent cases of ID theft are suspected to be the work of insiders. Either way, state-of-the-art security systems used by the financial industry have not been designed to prevent the transmission of sensitive information originating inside the organization.

What standard UNIX provides could be sufficient for protection from the outside world, but what if the inside user itself becomes malicious? In standard UNIX, you cannot stop an information leak if it is done by a malicious insider with access to the systems.

One approach to stop important information leaks is by classifying information at different levels and labeling them according to their level. You need to design a policy for the accessing information based on their different levels.

We can use an example of how can we use Trusted AIX for classifying information and labeling it accordingly and decide the policy that is good for stopping sensitive information leaks.

Health care industry

In the health care industry, we have information regarding the patient. We have different kinds of information that could be classified based on their sensitivity. For example, the previous treatments received by and the current treatment needed to the patient is sensitive information and should only be readable by the person with need-to-know. Information that is general, such as name and address, can be readable by other staff as well. So here is a classification of the information based on the sensitivity of the information.

For example, information that is needed to be readable *only* by doctors might be:

► Previous treatments received
► Current Treatment

Information needed to be readable *only* by a lab assistant and administrators might be:

► Medical test required

Information that should only be readable by the pharmacy department and administrators might be:

► Medicine
► Old Balance

Information that might be visible to all staff might be:

► Name
► Address
► Membership account number

Now that we have classified the information, we can label the information with different labels.

Let us assume that we have the labels and the mapping between the information and labels shown in Table 5-6.

Table 5-6   Example labels and information

Information	Sensitivity Label	
	Classifications	**Compartments**
1. History of the treatment received 2. Current Treatment	TREATMENTS	ENT,GYNO, ORTHO, OPTHAL, and ALL_DEP
3. Medical test required	RECORDS	
4. Medicine 5. Old Balance	MEDICINE	
6. Name 7. Address 8. Membership account number	GENERAL	

The different persons have been assigned the labels shown in Table 5-7.

Table 5-7   Example person assignment

Person	Classification	Compartment
Doctors	TREATMENTS	<Department Name>
Lab Assistant	RECORDS	<Department Name>
Medical store cum cashier	MEDICINE	ALL_DEP
Receptionist	GENERAL	ALL_DEP

Doctors will have the classification of TREATMENTS and the compartment as their department name. Lab assistants will also be specific to the departments and will have the classification of RECORDS and compartment as the department name they belong to. The pharmacy and the receptionist will belong to all the departments, so they will have the compartment as ALL_DEP, that is, all departments.

How classifying helps in this situation

► Confidentiality:

– Information leakage is not possible. Doctors cannot write to the lower level file and will not be able to leak the information to lower level people.

– Keeping the labels of the doctors as different compartments makes them access only the records related to their departments. They cannot access or modify the records for other departments.

– There could be one head doctor who will have the label TREATMENT ALL_DEP and can access all department records and can refer to a specific department doctor.

– A person working at lower level cannot access the files at a higher level. They cannot read the information that they are not supposed to read.

– The printer can be configured so that no one can print out confidential documents.

– Other external storage devices can be set at a high level so that the staff will not be able to write those documents to the external media.

► Integrity:

The lower level people cannot modify the file at upper levels based on the MAC policy.

► Accountability:

An enhanced Audit system can be configured for the Multilevel Security Attributes auditing to take care of accountability.

Managing system

Most of the system management is done by the users with *ISSO (information system security officer)* and *SA (System administrator)* roles.

The configuration tasks to be done by them are shown in Table 5-8.

Table 5-8   Configuration tasks

Configuration task	Who	OR
Label configuration	isso	A user with aix.mls.lef authorization
User creation	sa	A user with aix.security.user.create authorization
Assignment of labels	isso	A user with aix.mls.clear.write authorization
System configuration	isso	A user with aix.mls.system.config.write authorization
Printer configuration	isso	A user with aix.device.config.printer authorization
Network configuration	isso	A user with aix.mls.network.config and aix.mls.network.init authorization

Label configuration

To add the additional label to the Label Encoding file, we need to copy the Standard LEF file to some other file, edit that file, and then replace the standard LEF file with the modified file. This can be done by using the **labck** command, which can be run by an aix.mls.lef authorized user or an isso user:

```
$ labck -c temp_lef
```

Edit the file temp_lef to include the new classifications in the CLASSIFICATION section. We will be merging the new labels in between the existing label. You can replace the full API.

Example 5-28   Add the labels to the CLASSIFICATION section

```
**************************************************
VERSION= AIX VERSION
**************************************************


**************************************************
CLASSIFICATIONS:
**************************************************


        name= IMPLEMENTATION LOW;      sname= IMPL_LO; value= 0;
        name= GENERAL;                 sname= GEN;     valur= 3;
        name= MEDICINE;                sname= MED;     value= 5;
        name= RECORDS;                 sname= REC;     value= 10;
        name= TREATMENTS;              sname= TRET;    valure= 15;
```

As the values assigned to them are in increasing order, the dominance relationship will be same. That is, TREAMENTS will dominate RECORDS and RECORDS will dominate MEDICINE.

Now add the different compartments to the SENSITIVITY LABEL section under WORD: subsection (see Example 5-29). We have added the new compartments between the existing ones. Old ones can be removed, but we do not recommend this action.

> **Important:** Follow the MITRE standard to modify the file and check the file for correctness using the **labck** command before replacing the LEF file.

Example 5-29   Sensitivity Labels

```
**************************************************
SENSITIVITY LABELS:
**************************************************

WORDS:

        name= ALL;       sname= ALL;      compartments= 0-89;
        name= GIBRALTAR;sname= GIB;       compartments= 1-5;
        name= COMP_A;    sname= A;        compartments= 1;
        name= COMP_B;    sname= B;        compartments= 2;
        name= COMP_C;    sname= C;        compartments= 3;
        name= COMP_D;    sname= D;        compartments= 4;
        name= ENT;       sname= E;        compartments= 6;
        name= GYNO;      sname= G;        compartments= 7;
        name= ORTHO;     sname= O;        compartments= 8;
        name= OPTHAL;    sname= OP;       compartments= 9;
        name= ALL_DEP;   sname= AD;       compartments= 6-10
        name= INTERNET;  sname= INET;     compartments= 67;
```

Add the labels in Example 5-30 to the ACCREDITATION RANGE section.

Example 5-30   Add the new labels to ACCREDITATION RANGE section

```
**************************************************
ACCREDITATION RANGE:
**************************************************

        classification= IMPL_LO;        only valid compartment
combinations:

        impl_lo

        classification= GEN;    all compartment combinations valid;
        classification= MED;    all compartment combinations valid;
        classification= TRET;   all compartment combinations valid;
        classification= REC;    all compartment combinations valid;
```

After editing the file, we can replace the standard file with the modified one.

Important: Check that the Label Encoding File is valid after the modification using the **labck** command before replacing the Standard Label Encoding file:

```
$ labck -f temp_lef
```

Note: Take a backup of the original file before replacing it with the modified one:

```
$ labck -r temp_lef
```

System configuration

Enable or disable the services according to your needs. For enabling the Sensitivity Label checking, enable the sl flag with the **setsecconf** command, if it is not already enabled. The current setting can be checked by the **getsecconf** command.

Example 5-31   Enable the SL enforcements

```
$ setsecconf -o sl=enable
OPERATIONAL MODE Security Flags
TRUSTED NETWORK           :   DISABLED
SL ENFORCEMENT            :    ENABLED
TL WRITE ENFORCEMENT      :    ENABLED
TL READ ENFORCEMENT       :    ENABLED
TLIB                      :   DISABLED
ROOT                      :   DISABLED
```

User account creation

Only a user with the sa role can create the user. Log in as the user with the sa role and create the user using the **mkuser** command (Example 5-32).

Example 5-32   Create user as a user with sa role

```
$ su sa
$ swrole sa
sa's password:
$ mkuser doctENT
$ mkuser doctORTH
$ mkuser labENT
$ mkuser labORTH
$ mkuser medicALL
$ mkuser recept
```

We have created a name that could represent the person as the job he does and the department he is in (Table 5-9).

Table 5-9   Person/label assignment

Person	Labels assigned	Description
doctENT	TRET ENT	Doctor in ENT department
doctORTH	TRET ORTHO	Doctor in ORTHOPAEDICS department
labENT	REC ENT	Lab Assistant from ENT department
labORTH	REC ENT	Lab Assistant from ORTHOPAEDICS department
medicALL	MED ALL_DEP	Person form Medical store
recept	GEN ALL_DEP	The receptionist

Label assignment

Assigning a password to the users and assigning labels can only be done by the user with the ISSO role.

A label can be assigned to the users by the `chuser` command by the isso administrator (Example 5-33).

Example 5-33   Assign label to the users based on their job and department

```
$ chuser minsl="GEN ALL_DEP" defsl="GEN ALL_DEP"  maxsl="GEN ALL_DEP"
recept

$ chuser minsl="MED ALL_DEP" defsl="MED ALL_DEP"  maxsl="MED ALL_DEP"
medicALL

$ chuser minsl="REC ORTHO" defsl="REC ORTHO"  maxsl="REC ORTHO" labORTH

$ chuser minsl="REC ENT" defsl="REC ENT"  maxsl="REC ENT" labENT

$ chuser minsl="TRET ENT" defsl="TRET ENT"  maxsl="TRET ENT" doctENT

$ chuser minsl="TRET ORTHO" defsl="TRET ORTHO"  maxsl="TRET ORTHO"
doctORTH
```

Note: Change the home directory label of the users as well change them to the same level so that they can create files in that directory.

Creating and reading a file

1. doctENT logs in to the system and stores the information in a file treatment.

2. The file will be created at the same level (Example 5-34).

Example 5-34   User doctENT logs in to the system and creates a file with treatments

```
$ su - doctENT
$ lstxattr -p -F $$
430314:
        effsl=TREATMENTS ENT
        maxcl=TREATMENTS ENT
        mincl=TREATMENTS ENT
$ cat > treatment
Current Treatments
Previous treatments
$ lstxattr -f -F treatment
treatment:
        sl=TREATMENTS ENT
```

3. Now the doctors from the other departments and the other staff will not be able to read these files.

4. Log in as the receptionist and try to read that file (Example 5-35).

Example 5-35   Log in as the receptionist and try to read the file

```
$ su - recept
$ lstxattr -p -F $$
368672:
        effsl=GENERAL ENT GYNO ORTHO OPTHAL ALL_DEP
        maxcl=GENERAL ENT GYNO ORTHO OPTHAL ALL_DEP
        mincl=GENERAL ENT GYNO ORTHO OPTHAL ALL_DEP
$ cat /home/doctENT/treatment
cat: 0652-050 Cannot open /home/doctENT/treatment.
```

5. The person at the lower level will be able to read and write to the file at the same level.

Printer configuration

The administrator can configure the range of the labels for the printer. (See 5.5.8, "Printer configuration" on page 323). He can prevent users from printing the data on the printer (Figure 5-16).
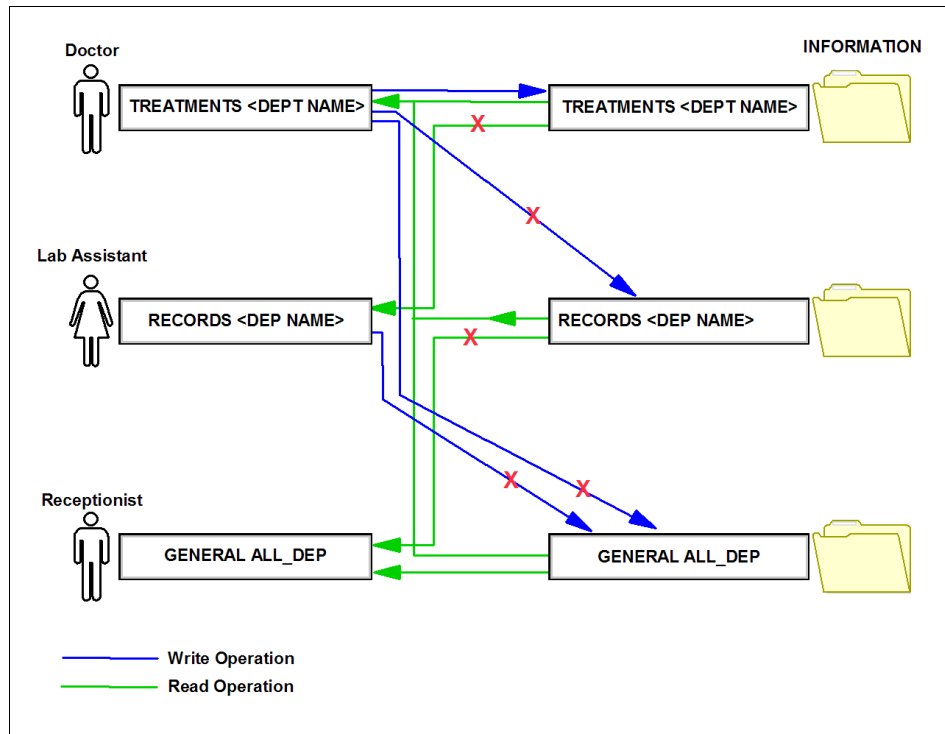
Figure 5-16   Pictorial view

Advantages

In UNIX, we can have groups and we can arrange our setup so that different groups cannot access the files of the other group. But what if we have a situation where we have people in the same group and we want to distinguish them from the rest of the group? Multi level security is more granular.

In standard UNIX, if the person changes the permission bits, intentionally or unintentionally, information could be readable to every one. In Multi Level Security, a user cannot change the level of his or her file. The security is controlled by the system.

A user operating at a higher label cannot write to the files in the lower level. He cannot leak the information by writing to the lower level files, which are readable by the person operating at lower level. The lower level person cannot read the files at a higher level, so he cannot read the information at higher level.

Because the printer and other external storage devices can be configured to be accessed only by people within specific label ranges, not everyone can print or write the data to the external device. In this way, information leaked by writing it to the external media or printing can be avoided.

5.7 Best practices and ideas

► We recommend that the system is always operated in *operational mode* so that all their services can be available.

► The maximum security level in the system's accreditation range should not be greater than the maximum security level for the site in which it is located.

► The system hardware should be in a secure location. The most secure places are generally interior rooms that are not on the ground floor.

► Physical access to the system hardware should be restricted, monitored, and documented.

► System backups and archival media should be stored in a secure location, one separate from the system hardware site. Physical access to the media should be restricted in the same manner as access to the system hardware.

► Access to operating manuals and administrative documentation should be restricted on a valid need-to-know basis.

► Unusual or unexpected behavior of any system software should be documented and reported, and its cause determined.

► Whenever possible, at least two individuals should administer a system. One should be assigned ISSO authorization and the other should be assigned SA authorization.

► The PV_ROOT privilege should not be used. To administer the system, the execution of privileged programs by ISSO, SA, or SO authorized users should suffice.

► Audit information should be collected into logs and reviewed regularly. Irregular or unusual events should be noted and their cause investigated.

► One user should not be assigned all three roles of ISSO, SA, and SO.

6

AIX Security Expert

This chapter describes the AIX Security Expert tool in AIX V6. It was originally introduced with AIX 5L V5.3 TL05 to provide a single point of administration for all security settings in AIX.

6.1 Introducing AIX Security Expert

Starting with AIX 5L V5.3 TL05, this system security hardening tool assists an AIX admin in having a single consistent view to all of security configuration and enablement, for example, TCP/IP, IPSec, system security settings, and auditing, just to name a few.

The admin does not have to be a security expert or maintain a huge collection of individual tools to bring a system to a required security level, keep it there, and to comply with regulations. AIX Security Expert can be run from Web-based System Manager, SMIT, or by using the `aixpert` command.

The features of AIX Security Expert that were introduced in AIX 5L V5.3 TL05, like setting various security levels at the click of one button or creating snapshots of a system's current security settings, have been extended to include new features, such as:

► Secure by Default installation

► Distributed security policy through AIX Security Expert and LDAP

► Customizable security policy through user defined AIX Security export XML rules and rule check

► The File Permission Manager `fpm` command for managing setuid and setgid programs

► Stringent check for weak passwords

► Secure File Transfer Protocol (ftp on TLS)

6.2 The next generation AIX Security Expert in AIX V6

Compared to the previous release of AIX Security Expert, much focus has been put on centralized policy distribution and compliance checking. The new options for both the CLI and GUI flavor of `aixpert` are illustrated in Example 6-1 on page 337.

Example 6-1   New AIX Security Expert options

```
# aixpert -?
aixpert -l high|medium|low|default|sox-cobit [-p] [ -n -o filename ] [
-a -o filename ]
aixpert -l h|m|l|a|d|c|s [-p] [ -n -o filename ] [ -a -o filename ]
aixpert -c [-l <level>] [-p]
aixpert -u [-p]
aixpert -d
aixpert [ -f filename ] [ -a -o filename ] [-p]
```

As in its previous version, the three security levels low, medium, and high still exist, as does the advanced option and the default settings (factory reset). For ease of use, however, the advanced settings have been removed from the CLI of `aixpert`. The GUI (either `wsm` or the Remote WebSM Client) is viewed as the best practices use case to security harden a single system. After the single system is configured with the GUI and tested, the AIX Security Expert is then used to distribute and apply these settings to other systems. This process is explained in detail later on.

The undo option (-u) no longer needs an XML file to be provided. All changes (with very little exceptions, which are explained later) made through aixpert will be saved in /etc/security/aixpert/core/undo.xml and can be undone any time. For example, `aixpert -u` will undo all actions that are stored in the undo.xml file, whereas when using the GUI you will be prompted first about which actions from the undo file you wish to undo. Some actions cannot be undone. For a complete list of these modifications, please refer to Table 1-4 on page 17.

Compliance checking against the active configuration is still done with the -c option, but it has been extended to check against a predefined level as well.

All options that would change the system settings now come with a preview mode (the -p option). That way the admin can run the command and see what is going to be changed without actually altering the system.

The GUI's main menu is shown in Figure 6-1.
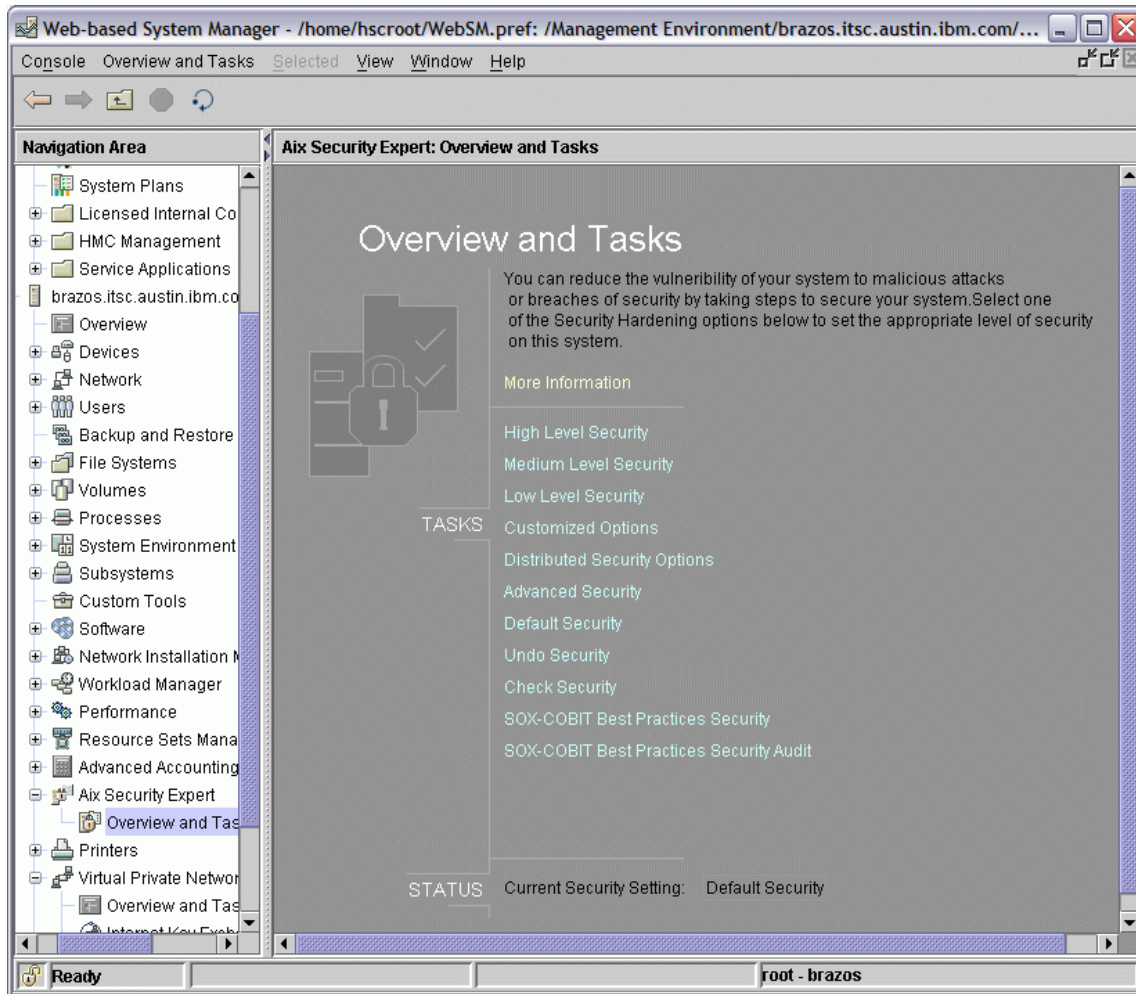
Figure 6-1   The AIX Security Expert Main Menu

All configuration data still resides in and under /etc/security/aixpert. Table 6-1 gives a rundown of AIX Security Expert's sub-directories.

Table 6-1   AIX Security Expert's directories

Directory name	Purpose
bin	All scripts (ksh) needed to perform AIX Security Expert's actions are stored in here. If needed, add your own scripts here as well.
core	Repository of XML files of all predefined levels and policies.
custom	If this directory does not exist, create it. It will be used to hold any self-defined policies. The GUI's "Customized Options" will check for any XML file in this directory and present it in a selection menu.
dictionary	Directory to hold the predefined English dictionary (can be any language, since it is just a word list) file for weak root password detection. Additional dictionaries should be saved in here as well.
ldap	If this directory does not exist, create it. The GUI's "Distributed Security Options" will check for any AIX Security Expert policy file available on an LDAP server, will download those XML files, and will place them in here for further usage.
log	Holds the aixpert.log file, a trail of all activities conducted by the AIX Security Expert.
tmp	Directory for temporary files that are needed during AIX Security Expert operations.
undo	This directory is used by the hardening scripts in bin to keep track of their undo operations. You might want to use it for self-written scripts as well.

6.3 Secure by Default (SbD)

Up to AIX 5L V5.3, there used to be two security related options that could be enabled at install time only: Trusted Computing Base (TCB) and CAPP/EAL4+ (CCEVAL). Starting with AIX V6, three new options have been introduced: Trusted AIX, LSPP/EAL4+, and Secure by Default (SbD). Like the existing ones in AIX 5L V5.3, these options can only be enabled at install time.

Attention: We use the abbreviation *SbD* for *Secure by Default* in many places throughout this chapter.

Trusted AIX (either in regular mode or in Common Criteria certified LSPP mode) introduces a completely new concept of running and administrating an AIX system. Trusted AIX, which is an Multi Level Security (MLS) system is described in Chapter 4, "Trusted Execution environment" on page 251.

Secure by Default takes a bottom-up approach in hardening an AIX system by installing a minimal set of software, because any additional software could potentially be exploited as a security vulnerability and then applying a "high security level" hardening to those components (Figure 6-2). This approach is opposite to starting with a regular, full-blown AIX installation and then use the AIX Security Expert to apply hardening (top-down approach) by disabling unneeded components.

> **Note:** Since a Secure by Default system is missing parts of the bos.net.tcp.{client|server} filesets, thorough planning and testing is required in order to fulfill all requirements your applications might have.
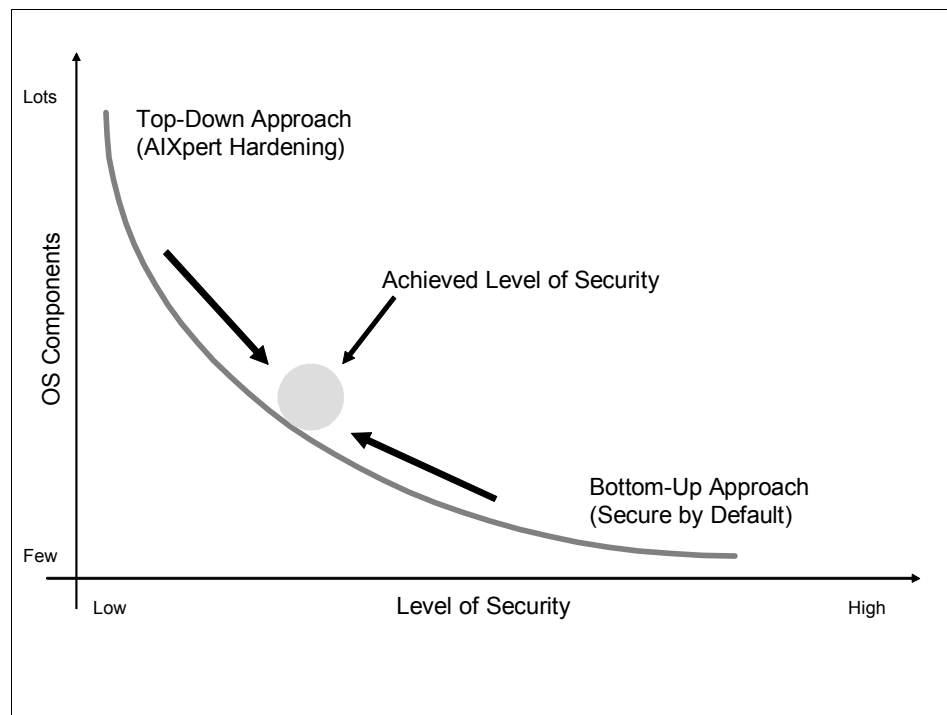
Figure 6-2   The Functionality versus Security Dilemma

6.3.1 Installing a Secure by Default system

When Secure by Default (SbD) is selected at install time, only a minimal AIX installation (approximately 100 filesets) occurs. This is significantly less than what would be installed during a regular, full-blown AIX installation (at least 250 filesets). The idea is that SbD provides a minimal but working AIX system to start. It is up to you and your requirements to add filesets later as needed.

Figure 6-3 through Figure 6-6 on page 343 show the screens you have to navigate through to start an SbD installation:

```
                    Installation and Settings

 Either type 0 and press Enter to install with current settings, or type the
 number of the setting you want to change and press Enter.

    1  System Settings:
         Method of Installation.............New and Complete Overwrite
         Disk Where You Want to Install.....hdisk0

    2  Primary Language Environment Settings (AFTER Install):
         Cultural Convention................English (United States)
         Language...........................English (United States)
         Keyboard...........................English (United States)
         Keyboard Type......................Default
    3  Security Model......................Default
    4  More Options  (Software install options)

>>> 0  Install with the settings listed above.


                        +----------------------------------------------------
    88  Help ?           |  WARNING: Base Operating System Installation will
    99  Previous Menu    |       destroy or impair recovery of ALL data on the
                         |             destination disk hdisk0.
>>> Choice [0]: 3
```
Figure 6-3   Main installation screen

```
                        Security Models

  Type the number of your choice and press Enter or
  press Enter to continue to next choices.

 1. Trusted AIX............................................. no

 2. Other Security Options  (Trusted AIX and Standard)
     Security options vary based on choices.
     LSPP, SbD, CAP/CCEVAL, TCB




>>> 0  Continue to more software options.

    88  Help ?
    99  Previous Menu

>>> Choice [0]: 2
```

Figure 6-4   Security Model install-time-only options

```
                      Standard Security Options

  Type the number of your choice and press Enter or
  press Enter to continue to next choices.

 1. Secure by Default........................................ yes

 2. CAPP and EAL4+ Configuration Install..................... no

 3. Trusted Computing Base Install........................... no






>>> 0  Continue to more software options.

    88  Help ?
    99  Previous Menu

>>> Choice [0]:
```

Figure 6-5   The Standard Security Options screen

```
                              Install Options




  1.   System Management Software for Secure by Default................. yes
       (Java, Websm)
  2.   Create JFS2 File Systems.......................................... Default










>>> 0  Install with the settings listed above.

    88  Help ?
    99  Previous Menu

>>> Choice [0]:
```

Figure 6-6   Final screen for Secure by Default installation

Secure by Default is a system state that gets saved in the ODM (just like TCB or
CCEVAL enabled). You can query this state later on a running system by
running:

odmget -q attribute=SbD_STATE PdAt

PdAt:
        uniquetype = ""
        attribute = "SbD_STATE"
        deflt = "sbd_enabled"
        values = ""
        width = ""
        type = ""
        generic = ""
        rep = ""
        nls_index = 0

During installation, this state is checked by the installation routines of the TCP client and server filesets. If an sbd_enabled system is detected, a number of binaries are removed because they are considered to be insecure (that is, they use very weak authorization only, if at all). From the bos.net.tcp.client fileset these are:

```
/usr/bin/rcp /usr/bin/rexec /usr/bin/rsh /usr/bin/remsh /usr/bin/tftp
/usr/bin/utftp /usr/lib/boot/tftp /usr/bin/traceroute
/usr/sbin/sendmail_nonssl /usr/sbin/sendmail_ssl /usr/sbin/sendmail
/usr/sbin/mailq /usr/sbin/newaliases /usr/lib/sendmail
/usr/sbin/sendmail_load /usr/sbin/netcd /usr/sbin/netcdctrl
/usr/samples/tcpip/netcd.conf /usr/lib/drivers/if_op
```

These files will be removed from the bos.net.tcp.server fileset:

```
/usr/sbin/gated /usr/sbin/imapd /usr/sbin/ipreport /usr/sbin/iptrace
/usr/sbin/named8 /usr/sbin/named8-xfer /usr/sbin/named9
/usr/sbin/pop3ds /usr/sbin/routed /usr/sbin/tcpdump /usr/sbin/timed
```

Once installation finishes and the system reboots, **aixpert** is executed once through /etc/firstboot using the predefined policy SbD.xml, that is, the command that gets actually executed is **aixpert -f /etc/security/aixpert/core/SbD.xml**. (This file is also your reference for what is changed on an SbD system in addition to the initial removal of binaries.)

After this initial hardening is completed, the reboot finishes like any regular AIX installation, that is, you get prompted by the Installation Assistant to apply some basic settings to the system (accept licences, set the date and time, set root password, and so on). Since the SbD policy is based on the high security level policy, strict password rules are already in place.

Adding filesets should be done from either a CD/DVD or in a secure and segregated networking environment to avoid exposure. We highly recommend that OpenSSH is installed right after the initial installation, because SbD will remove any remote services that use clear-text passwords (for example, **telnet**, **ftp**, **rlogin**, and so on). Any further customization can then be done remotely in a secure manner.

Note: If bos.net commands and functions like `telnet`, `ftp`, `rlogin`, and so on are required for your operating environment, then the SbD option is not for you. If the SbD install option is chosen, and at a later date it is determined that these functions are required for your operating environment, or newly installed filesets, then the SbD state of the system can be reset. Re-adding those binaries to the system and leaving it in the SbD state might result in undesired effects when verifying the system's security status through `aixpert` or when applying updates to the base operating system. See 6.3.2, "Reverting from Secure by Default back to regular AIX" on page 345 on how to reset the ODM entry.

6.3.2 Reverting from Secure by Default back to regular AIX

Any Secure by Default installed system can be turned back into a regular system later if you have the need to do so (for example, you need to add filesets from the bos.net package that were deleted by SbD). To revert back to regular AIX, follow these steps:

1. Run `odmget -q attribute=SbD_STATE PdAt > sbd_state`.
2. Edit the sbd_state file and change `sbd_enabled` to `sbd_disabled`.
3. Run `odmdelete -o PdAt -q attribute=SbD_STATE PdAt`.
4. Run `odmadd sbd_state`.
5. Re-install (with the force flag) bos.net.tcp.client and bos.net.tcp.server.
6. Run `aixpert -l d[efault]`.

6.4 Distributed security policy through AIX Security Expert and LDAP

In AIX V6, the AIX Security Expert offers a new method of distributing policy files from one system to another: LDAP. (At the time of writing, AIX V6 supported both LDAP Server V5.2 and V6.0.) When an LDAP server is available, policy files can be stored in the LDAP database. When applying policies on other systems, that LDAP server gets asked whether there are policy files available in the LDAP database. If XML files are found, they GUI will offer to load them onto the system.

We highly recommend that your LDAP server supports two types of LDAP clients. Only one system (or very few systems) should have admin rights on the LDAP server while the majority of clients should be allowed to just do their usual userID/PW housekeeping and the likes.

This kind of restriction is even more important when we are talking about security policies. You should only have a very few clients that are actually allowed to upload policies while the majority of clients just need to download these policies.

6.4.1 LDAP server preparation

After the successful installation of the LDAP server and the DB2® prerequisite filesets, you need to configure the system as an LDAP server. In order to be able to support the aforementioned two groups of clients, the LDAP server needs to be set up with a fully privileged admin CN and a less privileged proxy CN. Do so by simply using the `mksecldap` command (feel free to use whatever CNs you like):

```
# mksecldap -s -a cn=admin -p admin -S rfc2307aix -x
cn=aixpp,cn=aixdata -X aixpp
```

Of course you might as well use any existing LDAP server in your environment if it supports any schema AIX needs (that is, AIX, RFC2307, or RFC2307AIX).

Whatever LDAP server you will be using, the next thing to do is to extend the schema in order to be able to store your XML files in there. This is accomplished by the following command and the LDIF file, which will be included in the bos.aixpert.* filesets and can also be found in Appendix B, "LDIF file for supporting AIX Security Expert" on page 381:

```
# ldapmodify -D cn=admin -w admin -i IBM.V3.aixpert.schema.ldif
modifying entry cn=schema
```

One common pitfall of this command is when the LDIF file was not transferred correctly between different types of operating systems that use different CR/LF characters (as, for example Windows® and AIX). So, in case you run into those error messages:

```
modifying entry cn=schema
ldap_modify:  DSA is unwilling to perform
ldap_modify:  additional info: Invalid attribute type for schema entry
```

the LDIF file simply needs to be converted into the proper text format (either by transferring it again using ftp in ASCII mode or by any other tool of your choice).

The last step in LDAP server preparation is to restrict the permissions on policy files for the proxy CN. Create an LDIF file with these lines and save it to some name:

```
dn: cn=aixpp,cn=aixdata
changetype: modify
replace: aclentry
aclentry:
access-id:cn=aixpp,cn=aixdata:at.ibm-aixpertXmlConfigFile:grant:r

dn: cn=aixpp,cn=aixdata
changetype: modify
replace: aclpropagate
aclpropagate: TRUE
```

and add it to your server's schema by using:

```
# ldapmodify -D cn=admin -w admin -i <some_name>
```

Your LDAP server is now ready to store AIX Security Expert policy files and will support two bind IDs with different privileges. Of course, you can configure your LDAP server to be an LDAP client as well by using:

```
 mksecldap -c -h localhost -a <the privileged or unprivileged CN> -p
<the resp. password> -A ldap_auth.
```

> **Tip:** In case you ever need to start from scratch with your LDAP server configuration, make sure to drop the DB2 instance, because information about this will be preserved even across deinstallation and reinstallation of LDAP and DB2 filesets and deletion of the DB2 files directory.
>
> The proper steps for cleaning up any existing LDAP server configuration in order to start all over are (to be run as root):
>
> 1. On LDAPv5.2, **kill** the ibmslapd process; on LDAPv6, run **ibmslapd -k**.
>
> 2. Run **su - ldapdb2 -c db2 drop database ldapdb2**.
>
> 3. on LDAPv5.2, run **/usr/ldap/db2/instance/db2idrop ldapdb2**;
>   on LDAPv6, run **/usr/opt/db2_08_01/instance/db2idrop ldapdb2** and
>   **idsidrop -nI ldapdb2**.
>
> 4. Run **mksecldap -s -U**.
>
> 5. Run **rmuser -p ldapdb2**.
>
> 6. Run **rm -rf /home/ldapdb2**.

6.4.2 LDAP client preparation

In order to be able to upload policies to your LDAP server, the system you are about to harden needs to be configured as an LDAP client first. Install either the ldap.client.* filesets for LDAP Server V5.2 or the idsldap.clt32bit* filesets for LDAP Server V6.0.

As mentioned in 6.4.1, "LDAP server preparation" on page 346, we want to be able to restrict the ability to upload policies to a very few clients only while providing general read access to all policies for any other client.

Setting up an LDAP client with admin rights

You can configure the system to run as a privileged LDAP client.

To configure the system as a privileged LDAP client, run the following command:

```
mksecldap -c -h <ldap_server> -a cn=admin -p admin -A ldap_auth
```

You should use ldap_auth in order for the ACLs to work properly.

Use either **aixpertldap** to upload and download policies or the GUI to download only.

We provide some examples showing how to use **aixpertldap** on an admin client. Example 6-2 shows how to upload the current configuration under the system's name. In Example 6-2, /etc/security/aixpert/core/appliedaixpert.xml will get uploaded to the LDAP DB.

Example 6-2   aixpertldap upload

```
# aixpertldap -u
adding new entry ibm-aixpertLabel=brazos,ou=aixpert,cn=aixdata
aixpertldap.sh: successfully uploaded XML config file on brazldap
```

Example 6-3 shows how to upload the file given the -f option and store it under the name given the -l option in the LDAP DB. Please note that when uploading any XML file, it must be given its complete path name starting with a slash (/). After the two policies have been successfully uploaded to the server, click **Distributed Security Options** to bring up a window like that shown in Figure 6-7.

Example 6-3   aixpertldap upload with options

```
# aixpertldap -u -f /path/to/mypolicy.xml  -l mysec
adding new entry ibm-aixpertLabel=mysec,ou=aixpert,cn=aixdata
aixpertldap.sh: successfully uploaded XML config file on brazldap
```
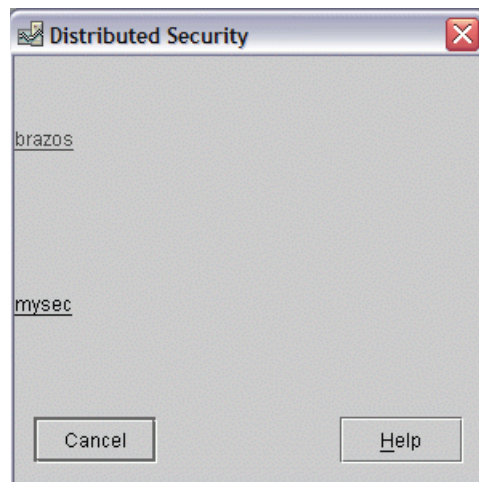

Figure 6-7   Distributed security policy selection window

In addition, all available policies have already been downloaded into the querying client's /etc/security/aixpert/ldap directory. They then can either be activated from the GUI or by using the following command in the CLI:

```
aixpert -f /etc/security/aixpert/ldap/<policy>
```

Setting up an LDAP client with restricted rights

To configure the system as a restricted LDAP client, run the following command:

```
mksecldap -c -h <ldap_server> -a cn=aixpp,cn=aixdata -p aixpp -A
ldap_auth
```

You should use ldap_auth in order for the ACLs to work properly.

Any client using our proxy CN to connect to the LDAP server will be able to manage its user IDs and passwords as usual, but will have read-only access to uploaded security policies.

Use either **aixpertldap** or the GUI to download policies. If you try to upload a policy from such a client, it will now result in an insufficient permissions error:

```
ldap_add:  Insufficient access
aixpertldap.sh: ldapadd command failed with code 50
```

6.5 Customizable security policy through user defined AIX Security Expert XML rules

It is possible to define your own security policy or rules that will automatically be integrated into the AIX Security Expert tool and GUI. Therefore, any security configuration policies unique to your environment or relating to third-party software, can be easily brought under the control and management of **aixpert**.

All security settings controlled by the AIX Security Expert are stored in XML format. The files shipping with AIX contain the settings for the predefined security levels high, medium, and low, as well as settings that assist in compliance with the SOX/COBIT standards.

One way to set up your own, customized policy might be to use the Advanced menu in the GUI. When selecting this menu, a window will pop up showing all available options from the /etc/security/aixpert/core/aixpertall.xml file. Figure 6-8 shows a snapshot of the first few options.
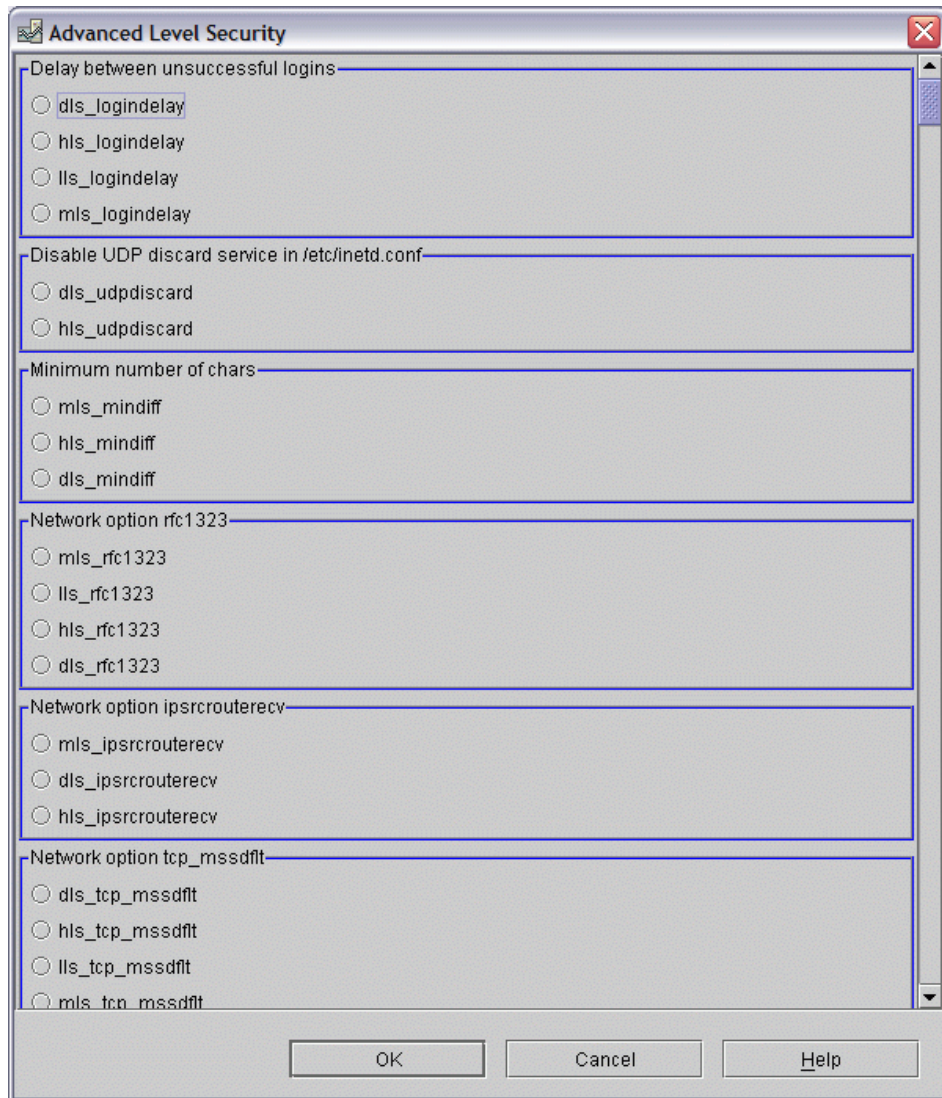
Figure 6-8   Some options from the Advanced Level Security Menu

You might as well use any text editor to create your own XML file from scratch or use parts of existing XML files and combine them to meet your specific requirements. Either way you do it, once you are finished, place your policy file into /etc/security/aixpert/custom/<your_policy>.xml. The GUI will automatically discover and display these XLM files as user selectable options in the Custom Security Files window.

So, in our example, when you click **Custom Options** in the main menu, you will see a window popping up, presenting all your customized policy files for selection, as shown in Figure 6-9.
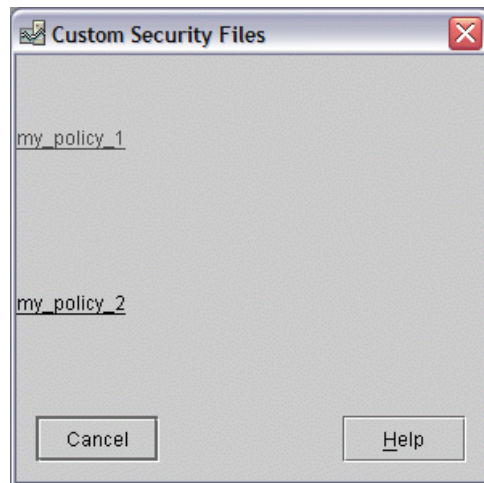
Figure 6-9   Custom security policy selection window

6.5.1 Adding rules for your own applications

When OEM products get installed in AIX, they can be added to AIX Security Expert in order to get managed by this tool as well.

In order to add components to the AIX Security Expert database or to modify existing profiles, this simply needs to be coded into the XML files. The XML DTD (Document Type Declaration) is shown in Example 6-4.

Example 6-4   XML DTD for AIX Security Expert Database

```
<?xml version='1.0'?>
<!--START-->
<!ELEMENT AIXPertSecurityHardening (AIXPertEntry+)>
<!-- AIXPertEntry should contain only one instance of the following
     elements.
-->
<!ELEMENT AIXPertEntry (AIXPertRuleType,
    AIXPertDescription, AIXPertPrereqList, AIXPertCommand,
    AIXPertArgs,AIXPertGroup)
>

<!-- AIXPertEntry's name should be unique.
-->

<!ATTLIST AIXPertEntry
    name ID #REQUIRED
    function CDATA ""
>

<!ELEMENT AIXPertRuleType EMPTY>
<!ATTLIST AIXPertRuleType
    type (LLS|MLS|HLS|DLS|SCBPS|Prereq) "DLS"
>
<!ELEMENT AIXPertDescription (#PCDATA)>
<!-- the AIXPertDescription tag will allow you in insert a detailed description
of this option. When the admin views this custom configuration and places their
mouse over it, this detailed description will automatically be displayed in a
pop up information window -->

<!ELEMENT AIXPertPrereqList (#PCDATA)>
<!-- The AIXPertPrereqList tag allows this particular entry to reference a
prereq entry. If a prereq is listed here, aixpert will run the prereq rule
prior to displaying or running this rule. The prereq list must consist of the
unique name, AIXPertEntry tag, of the prereq.  This is how aixpert finds and
executes the prereq rule. If the prereq rule returns "fail" (i.e. returns a
non-zero value) then this rule will not be selectable or run-able by aixpert.
Also on prereq failure, the aixpert GUI will grayout this rule and display the
description of the prereq rule, therefore, the prereq rule description should
describe how to resolve the prereq rule, i.e. "install openssh.base.server" -->

<!ELEMENT AIXPertCommand (#PCDATA)>
<!-- AIXPertCommand tag is the full path and command which aixpert will execute
-->
```

```
<!ELEMENT AIXPertArgs (#PCDATA)*>
<!-- The AIXPertArgs tag is all arguments which aixpert will apply to the above
command -->

<!ELEMENT AIXPertGroup (#PCDATA)*>
```

Let us, for example, include OpenSSH to the AIXpert database to be started in selected security levels. The AIXPertEntry name needs to be unique in your XML file; the function can be anything meaningful to you. Such an entry in XML could look like what is provided in Example 6-5.

Example 6-5   Definition example of a custom rule

```
<?xml version="1.0" encoding="UTF-8"?>

<AIXPertSecurityHardening>
  <AIXPertEntry name="xn_sshd" function="sshd">
    <AIXPertRuleType type="LLS"/>
    <AIXPertDescription>Activate OpenSSH, if
installed.</AIXPertDescription>
    <AIXPertPrereqList>openssh.base.server</AIXPertPrereqList>
    <AIXPertCommand>/usr/bin/startsrc</AIXPertCommand>
    <AIXPertArgs>-s sshd</AIXPertArgs>
    <AIXPertGroup>Start secure daemons</AIXPertGroup>
  </AIXPertEntry>
</AIXPertSecurityHardening>
```

When this file is placed in /etc/security/aixpert/custom, it can be either activated by **aixpert -f mypol.xml** from that directory or be selected from the GUI, where all options will be offered in a subsequent window (in Figure 6-10, it is only one option).
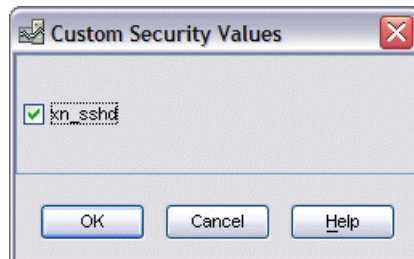
Figure 6-10   Example window of customized rule

When the script successfully completes (by having sshd activated), the status detail window will appear as shown in Figure 6-11.
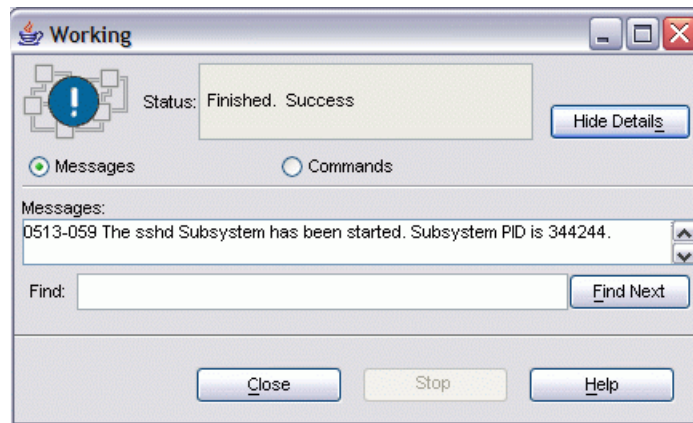
Figure 6-11   Result details of customized rule

6.5.2 The predefined SOX-COBIT security policy

The IT industry often uses COBIT IT Governance (Control Objectives for Information and related Technology, which is a set of best practices for IT management created by the Information Systems Audit and Control Association (ISACA) and the IT Governance Institute (ITGI) in 1992) as the *de facto* standard to comply with the Internal Controls Section 404 of the Sarbanes-Oxley[1] (SOX) act of 2002 (a United States federal law passed in response to a number of major corporate and accounting scandals). It is a focus of the AIX Security Expert to aid the admin in system compliance configuration and auditing.

Policies to common standards and best practices in IT security can be defined in XML to check for as many settings as possible. Currently, AIX Security Expert only ships with one predefined policy, but policy enhancements are envisioned in future releases. AIX V6 comes with one predefined policy for assisting you in SOX-COBIT compliancy setups and checks.

The SOX-COBIT policy of the AIX Security Expert is enforced by rules defined in /etc/security/aixpert/core/aixpertall.xml starting with SCBPS. At the time of writing, these rules trigger four scripts:

► /etc/security/aixpert/bin/pwdpolicyenf: Sets password quality rules.
► /etc/security/aixpert/bin/virusdetsw: Runs integrity checks of system binaries.
► /etc/security/aixpert/bin/firewsetup: Sets up shunning port filter rules.

---

[1] Pub. L. No. 107-204, 116 Stat. 745, also known as the Public Company Accounting Reform and Investor Protection Act of 2002 and commonly called SOX or Sarbox; July 30, 2002.

► /etc/security/aixpert/bin/secactreport: Creates a status report ## difference to "SOX/COBIT Audit"?

These files can be separately selected form the GUI when clicking SOX-COBIT Best Practices Security, as shown in Figure 6-12.

Figure 6-12   The SOX/COBIT Rules menu

The menu SOX-COBIT Best Practices Security Audit will run a check to see whether the current settings still match the ones defined in these script and will print a report (as shown in Figure 6-13). The aim of this report is to have a quick and easy statement of the system's current compliance status to be reported to some auditor, for example.
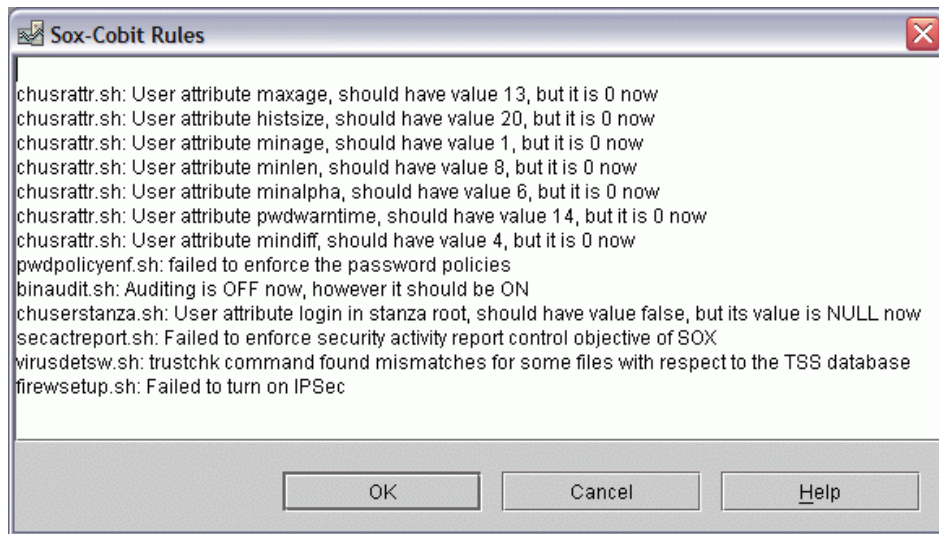
Figure 6-13 Screen capture of a SOX/COBIT audit

6.6 File Permission Manager for managing setuid and setgid programs

AIX has a long history of having many setuid and setgid programs. This used to cause concern to admins and auditors when using platform independent security tools that did not know much about this peculiarity. While releases prior to AIX V6 demand those s-bits (both setuid and setgid) in order to make TCB, earlier roles, and the auditing subsystem work correctly, the role based access control (RBAC) introduced with AIX V6 remedies this situation.

The File Permission Manager (`fpm`) will help you to cut down the number of setuid and setgid files whether you will be using RBAC or not. IBM has put quite some effort in investigating which files do need s-bits and which ones get along without. Each command affected by `fpm` was reviewed by IBM to determine if it was commonly used by the general user, or if its use was more specific to admin functions. If it was determined that a command's primary use was by admins, then `fpm` will turn off the setuid permission.

Use of the **fpm** command will limit the functionality of AIX to non-privileged users. Since **fpm** is a virtually self-containing tool, it will also become available for both AIX 5L V5.2 and 5L V 5.3, which do not offer RBAC.

The **fpm** command offers:

► Removing the setuid or setgid permissions from executables owned by privileged users, as well as any other non-privileged files to address the specific needs of unique computer environments.

► Logging the permission state of the files prior to changing them. The **fpm** log files are created in /var/security/fpm/log/<date><time>. If necessary, these log files can be used to restore the system's file permissions recorded in a previously saved log file.

> **Note 1:** Run thorough tests on your system after removing s-bits from binaries, especially user-created scripts that are used by non-privileged users and are calling those binaries that might not work any longer as expected. The **fpm** command provides the capability to restore the original AIX install default permissions (or undo any previous, specific change) using the -l default flag (Example 6-6).

Example 6-6   Restoring default permissions using fpm

```
# chmod 555 /usr/bin/acctctl
# ls -l /usr/bin/acctctl
-r-xr-xr-x    1 root     adm           201032 Apr 30 10:26 /usr/bin/acctctl
# fpm -l default
fpm will restore the AIX file permissions to the installed settings and any
customized defaults listed in /usr/lib/security/fpm/custom/default. If you had
done other customizations outside fpm and wish to return the file permissions
to a state representing a particular time and date, use the command:
 fpm -l default -f /var/security/fpm/log/<in_file>
Where <in_file> is a previously saved timestamped file representing this
system's file permission state at a particular date and time.
# ls -l /usr/bin/acctctl
-r-sr-s---    1 root     adm           201032 Apr 30 10:26 /usr/bin/acctctl
```

Example 6-7   The fmp command and extended permissions

```
# ls -l /usr/bin/vmstat
-r-sr-xr-x    1 root     bin              63394 Apr 30 10:00 /usr/bin/vmstat
# aclget /usr/bin/acctctl
*
* ACL_type   AIXC
*
attributes: SUID SGID
base permissions
    owner(root):  r-x
    group(adm):   r-x
    others:  ---
extended permissions
    enabled
    permit   r-x     u:foo
# fpm -l default
[fpm's default output deleted]
# aclget /usr/bin/acctctl
*
* ACL_type   AIXC
*
attributes: SUID SGID
base permissions
    owner(root):  r-x
    group(adm):   r-x
    others:  ---
extended permissions
    disabled
    permit   r-x     u:foo
```

Usage of the **fpm** command is pretty much straight forward and self-explaining, as you can see in Example 6-8.

Example 6-8   fpm usage

```
# fpm -?
Usage:  fpm -l <level>   | -f <file>  [-c ] [-v ] [-p] [-q] [-?]
 Where <level> = high | medium | low | default
  -l <level> determines the security level, where <level> may be one of the following:
  'high' - High level security. Removes suid and sgid permission from files listed in
/usr/lib/security/fpm/data/high_fpm_list.
  'medium' - Mediem level security. Removes suid and sgid permission from files listed in
/usr/lib/security/fpm/data/med_fpm_list.
  'low' - Low level security. Removes only the suid permission from files listed in
/usr/lib/security/fpm/data/med_fpm_list.
  'default' - Default level security returns file permissions to AIX default settings.
 -p passive mode prints out what changes would be made, but takes no action
  -s - Status of the current setting is displayed.  More precisely this flag displays the last
actionable running of the fpm command.
  -f <file> - Where <file> is a filename containing a list of files from which the permissions
should be removed.  The command will use this file instead of the default file list when
specifying -l <level> of security.  This option must be used in conjunction with -l <level>
option.
  -c - Checks that the files have the correct permission bits in accordance with a particular
level, but takes no action.  This option must be used in conjunction with -l <level>.
  -v - Verbose output.
  -q - quiet or minimal output.
  -? - Prints this usage statement.
```

The **fpm** command also offers a passive (or preview) mode. That way an admin gets an overview of what is going to be changed first, as shown in Example 6-9.

Example 6-9   fpm output in passive mode

```
# fpm -l low -p
chmod 0555 /opt/IBMinvscout/bin/invscoutClient_VPD_Survey
chmod 0555 /opt/IBMinvscout/bin/invscoutClient_PartitionID
chmod 0550 /usr/lpp/diagnostics/bin/diagsetrto
chmod 0550 /usr/lpp/diagnostics/bin/Dctrl
chmod 0550 /usr/lpp/diagnostics/bin/diagTasksWebSM
One or more file is already secure.  Therefore, the current file permissions may not
match the default permissions.  If you wish to return to the snapshot of permissions
prior to running this command, then use the command:
 /usr/bin/fpm -l default -f /var/security/fpm/log/04232007_11:57:36

 fpm will now continue to remove the SUID permissions.
chmod 0550 /usr/lpp/diagnostics/bin/diagela_exec
chmod 0555 /usr/lpp/diagnostics/bin/diagrpt
chmod 0550 /usr/lpp/diagnostics/bin/diagrto
chmod 0550 /usr/lpp/diagnostics/bin/diaggetrto
[...]
```

6.7 Stringent check for weak passwords

In the medium level and high level security policy, AIX Security Expert offers an option to check for weak root passwords (Figure 6-14).
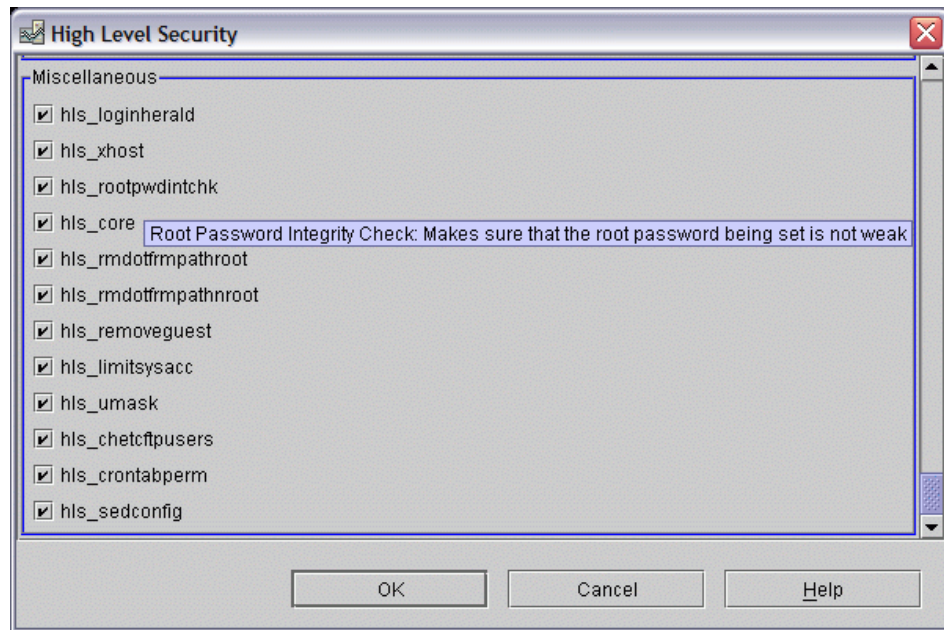
Figure 6-14   Root password strength check

This option will not check any existing root password, but it will enable the dictionary check in root's user stanza in /etc/security/user in order to accept only passwords not listed in the dictionary whenever the existing one is changed the next time.

Example 6-10   Checking for weak root passwords

```
root:
        admin = true
        SYSTEM = "compat"
        registry = files
        loginretries = 0
        account_locked = false
        dictionlist = /etc/security/aixpert/dictionary/English
```

This check guards against the use of English dictionary words and the 1000 most common US first names according to a recent US Census.

6.7.1 Adding entries to the dictionary

Since the AIX Security Expert simply uses the dictionlist option of the root user's stanza, it is very simple to add your own entries to the dictionary. Any dictionary file needs to be formatted as one word per line. The word begins in the first column and terminates with a new-line character. Only 7-bit ASCII words are supported for passwords in such dictionary files.

Depending on your requirements, you can either add your own set of weak passwords to the existing file or create a new file from scratch.

When adding words to an existing dictionary, you can simply append your entries to the file. The dictionary file does not have to be sorted. (It can even include an empty line that would catch null passwords, just in case such would slip through all other checking rules.)

If you would like to create a completely new dictionary, you might copy the existing one that ships with AIX Security Expert to a new file name and name your new dictionary English in path /etc/security/aixpert/dictionary/. That way no further modification to the XML database needs to be done.

If you would like to create a new dictionary and add it to the XML database to become available as an additional option, it is done the same way as you would set up a rule of your own (which is shown in 6.5.1, "Adding rules for your own applications" on page 352). The script that gets called to make this change is called chuserstanza in /etc/security/aixpert/bin.

For example, if your new dictionary is called Special and resides in /etc/security/aixpert/dictionary/, the stanza you have to add to /etc/security/aixpert/core/aixpertall.xml, in order to have this additional option available at the high level security settings, would need to be similar to what is shown in Example 6-11.

Example 6-11 Sample stanza to define another dictionary for password checks

```
<AIXPertEntry name="hls_rootpwdintchk2" function="rootpwdintchk">
    <AIXPertRuleType type="HLS"/>
    <AIXPertDescription>Root Password Integrity Check: Makes sure that
the root password being set is not weak (special
dict)</AIXPertDescription>

<AIXPertPrereqList>bos.rte.date,bos.rte.commands,bos.rte.security,bos.r
te.sh
ell,bos.rte.ILS</AIXPertPrereqList>

<AIXPertCommand>/etc/security/aixpert/bin/chuserstanza</AIXPertCommand>
    <AIXPertArgs>/etc/security/user
dictionlist=/etc/security/aixpert/dictionary
/Special root hls_rootpwdintchk</AIXPertArgs>
    <AIXPertGroup>Miscellaneous Rules</AIXPertGroup>
  </AIXPertEntry>
```

Note: If you want to have more than one dictionary checked, you need to modify the <AIXPertArgs> element to have dictionlist set to a comma separated list of all desired dictionaries.

6.8 Secure File Transfer Protocol

AIX V6 introduces a secure flavor of `ftp` (and `ftpd`), based on OpenSSL, using Transport Layer Security (TLS) (This extension to FTP is defined in RFC 4217.) to encrypt both the command and the data channel. TLS is a cryptographic protocol that provides secure communication between clients and servers.This enables any user on the system to exchange files in a secure manner if their counterpart offers this extension as well.

While on first sight using secure ftp only and no secure telnet might not be the most desirable option, this method is in fact a sensible alternative for environments where you are not able to use OpenSSH. For example, if your most trusted systems run on a dedicated and segregated network, it makes perfect sense to use telnet for remote access within that very network zone (or working from the console).

But even in such scenarios, you might very well have the need to transfer data from or to this secure zone, which can be accomplished now by using secure ftp.

Another scenario might be when you use OpenSSH already but you still have to exchange data with outside systems that do not support any form of SSH (either `scp` or `sftp`). Most often, such systems offer "ftp through ssl" (often called `ftps`) instead.

Since TLS relies on Secure Sockets Layer, make sure OpenSSL is installed on your AIX system (`ftp -s` depends on libssl.a and libcrypto.a). At least Version openssl-0.9.7l-1.aix5.1.ppc.rpm needs to be installed, which can be found on the AIX Toolbox for Linux Applications CD or on the respective IBM web site on the AIX Toolbox Cryptographic Content page:

http://www6.software.ibm.com/dl/aixtbx/aixtbx-p

Example 6-12 shows the command for checking the version of OpenSSL that is installed on your system.

Example 6-12   checking OpenSSL version

```
# lslpp -L "openssl*"
  Fileset                     Level  State  Type  Description (Uninstaller)
  --------------------------------------------------------------------------
  openssl                     0.9.7l-1   C    R    Secure Sockets Layer and
                                                   cryptography libraries and
                                                   tools (/bin/rpm)
```

If OpenSSL is not installed on your system or an older version is installed, you will need to install the required version.

6.8.1 Setting up ftpd to use TLS

Before any client can connect to your ftp daemon using TLS, a few preparations need to be made. If you already have a running Certification Authority (CA) in your environment, simply create your server certificates and copy them onto the systems you want to use encrypted FTP with. Set the server's /etc/ftpd.cnf file accordingly and you are all set. Do so according with your CA's public key, copy this one onto all your clients to let them know about your CA (in their ~/.ftpcnf

files) and complete the chain of trust. (See "Using self-signed certificates" on page 366 on how to set up the client and server configuration files correctly.)

Both openssl 0.9.71-1 and ftp/ftpd that ship with AIX V6 support V1 and V3 type X.509 certificates including certificate revocation lists (CRLs). However, for the sake of simplicity, the following sections will describe steps for two scenarios:

► Set up a simple V1 root CA to create self-signed certificates (without CRLs).

    and

► Import commercially signed certificates (v3) to be used with secure ftp.

The keys and certificates need to be present on all ftp servers you wish to exchange encrypted data with.

Using self-signed certificates

In this example, we will store all TLS relevant keys and certificates in root's ~/.tls directory on our server, but you can pick your own location if you wish. You simply need to change /etc/ftpd.cnf later to reflect the actual paths for the keys and certificates after completion.

1. Setting up the directory structure on the first server. This one will also have the CA keys and root certificate stored:

```
# cd
# mkdir .tls
# cd .tls
# mkdir rootCA
# chmod 700 rootCA
# cd rootCA
```

2. Creating a root level private key and root level certificate request (holding the public key):

```
# openssl req -newkey rsa:2048 -sha1 -keyout root_key.pem -out
root_req.pem
Generating a 2048 bit RSA private key
....................................................................
....................................................................
.................................................+++
..+++
writing new private key to 'root_key.pem'
Enter PEM pass phrase:<type anything here, at least 4 chars>
Verifying - Enter PEM pass phrase:<repeat the above>
-----
You are about to be asked to enter information that will be
incorporated into your certificate request.
```

```
What you are about to enter is what is called a Distinguished Name
or a DN.
There are quite a few fields but you can leave some blank.
For some fields there will be a default value.
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [US]:
State or Province Name (full name) [Some-State]:TX
Locality Name (eg, city) []:Austin
Organization Name (eg, company) [Internet Widgits Pty Ltd]:IBM
Organizational Unit Name (eg, section) []:CA
Common Name (eg, YOUR name) []:
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

You must enter a PEM pass phrase in order to protect your private root key. You should also enter some data for at least the first five fields in order to create a complete DN. Using less entries will result in certificates that will not work. (Please note that depending on the entropy on your system, the progress indicator will probably look different.)

3. Generating the certificate for root (valid approximately 10 years) by self-signing it:

```
# openssl x509 -req -days 3650 -in root_req.pem -signkey
root_key.pem -out root_cert.pem
Signature ok
subject=/C=US/ST=TX/L=Austin/O=IBM/OU=ITSO
Getting Private key
Enter pass phrase for root_key.pem: <enter your PEM pass phrase from
step 2>
```

You can have a look at your root certificate just to make sure everything is right by using:

```
# openssl x509 -in root_cert.pem -text -noout
Certificate:
    Data:
        Version: 1 (0x0)
        Serial Number:
            d2:01:13:b6:2d:b3:a8:b8
        Signature Algorithm: md5WithRSAEncryption
        Issuer: C=US, ST=TX, L=Austin, O=IBM, OU=CA
        Validity
            Not Before: Apr 26 19:45:52 2007 GMT
            Not After : May 23 19:45:52 2017 GMT
        Subject: C=US, ST=TX, L=Austin, O=IBM, OU=CA
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (2048 bit)
                Modulus (2048 bit):
                    00:97:57:11:84:e5:bb:a7:21:06:36:5b:1f:7b:b7:
                    [...]
```

When things look fine, you are finished with setting up your own root CA. We go up one directory level to create the first server key and certificate:

```
# cd ..
```

4. Now we are creating an RSA key for the first FTP server without a PEM pass phrase, hence we use a different command than the one we used in step 2 to create a new key:

```
# openssl genrsa 2048 > server_key.pem
Generating RSA private key, 2048 bit long modulus
...........+++
.........................................................
.........................................+++
e is 65537 (0x10001)
```

It is important not to use any pass phrases on such server keys. Otherwise, it would be required to input that pass phrase every time the key gets used (which is impossible to accomplish when ftpd is using it).

5. Next, we are creating a certificate request for the key we have just created (including its public key):

```
# openssl req -new -key server_key.pem -out server_req.pem
You are about to be asked to enter information that will be
incorporated into your certificate request.
What you are about to enter is what is called a Distinguished Name
or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [US]:
State or Province Name (full name) [Some-State]:TX
Locality Name (eg, city) []:Austin
Organization Name (eg, company) [Internet Widgits Pty Ltd]:IBM
Organizational Unit Name (eg, section) []:ITSO
Common Name (eg, YOUR name) []:
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

6. Next, we are signing the server key request with our root CA's private and self-signed public key. This will create the server certificate (again, this is valid for approximately 10 years):

```
# openssl x509 -req -days 3650 -in server_req.pem -CA
rootCA/root_cert.pem -CAkey rootCA/root_key.pem -CAcreateserial -out
server_cert.pem
Signature ok
subject=/C=US/ST=TX/L=Austin/O=IBM/OU=ITSO
Getting CA Private Key
Enter pass phrase for rootCA/root_key.pem: <enter your PEM pass
phrase from step 2>
```

7. In order to make server configurations easier as well as the distribution of certified key files, it is handy to have the server key, the server certificate, and the root certificate in one single file (OpenSSL supports this). So we are combining all three files to one file now:

```
# cat server_key.pem server_cert.pem rootCA/root_cert.pem >
server.pem
```

This file should be protected with respective file permissions to be accessible by root only (for example, 600). It can be copied to any other FTP server or you can repeat steps 4 through 7 for any additional FTP server you want to have its own signed key.

8. Finally, we adjust the path names in /etc/ftpd.cnf file:

```
CERTIFICATE      /root/.tls/server.pem
CERTIFICATE_PRIVATE_KEY /root/.tls/server.pem
```

Since we have combined all the keys and certificates in one file, we use that name for both the certificate and the key. Depending on your individual setup, this might be different if you are using separate files. All other lines must be left as comments. They are not needed in this simple self-signed rootCA scenario.

Using commercially signed certificates

If you are using commercially signed certificates already that can also be used for SSL (that is, if this is within their scope / key use), you can easily import them into your secure ftp setup. The following example uses certificates in privacy enhanced mail (PEM) security certificate format, but OpenSSL is supporting many other certificate formats (for example, PKCS#12) and offers easy to use conversion schemes for those (for example, http://gagravarr.org/writing/openssl-certs/general.shtml#cert-convert) as well as the option to remove any associated pass phrases.

1. Usually, you start with a private and public key pair. The private key needs to be referred to in /etc/ftpd.cnf by the CERTIFICATE_PRIVATE_KEY option.

2. Next, you get your public key signed by some trusted CA. This signed public key (along with the CA's root certificate) will become your server's certificate, which needs to be referred to in /etc/ftpd.cnf by the CERTIFICATE option.

In order to complete the chain of trust, all clients need to know all root CAs' certificates as well. Those can be downloaded from the Internet (http://www.columbia.edu/~ariel/good-certs/ is one of many places to start looking for CAs) and need to be referred to in any client user's ~/.ftpcnf by the CA_PATH option. (See also 6.8.2, "Setting up ftp to use TLS" on page 371.)

6.8.2 Setting up ftp to use TLS

You can use **/usr/bin/ftp** to connect to the FTP daemon on any remote system, whether this target host is TLS-enabled or not. Once you have completed the TLS setup on your server, in order to transfer files to or from that server, simply issue:

```
# ftp -s remote_host
```

This will establish an encrypted session, as shown in Example 6-13, without any modifications needed on the client side. If the certificate however is self-signed, the client has to blindly trust the server, hence the **ftp** command will display the most important data of the certificate it received during TLS handshake to leave the decision up to the user whether they want to connect or not.

Example 6-13   Encrypted ftp session using a self-signed certificate

```
# ftp -s nimrod
Connected to nimrod.
220 nimrod FTP server (Version 4.2 Mon Apr 9 11:38:07 CDT 2007) ready.
234 Using authentication type TLSv1
TLS Auth Entered.
TLS handshake succeeded, though Server signed it's own cert!
Certificate:
    Data:
        Version: 1 (0x0)
        Serial Number:
            ca:6d:1c:d0:7b:8d:ad:ea
        Issuer: C=US, ST=TX, L=Austin, O=IBM, OU=CA
        Validity
            Not Before: Apr 27 02:09:17 2007 GMT
            Not After : Apr 24 02:09:17 2017 GMT
        Subject: C=US, ST=TX, L=Austin, O=IBM, OU=Server
TLSv1/SSLv3 ( DHE-RSA-AES256-SHA ), 256 bits
Name (nimrod:root):
```

There is one further step of security by means of trust that can be established with such self-signed certificates: The rootCA's certificate needs to be copied to every client and referred to in their ~/.ftpcnt files of any user wishing to use encrypted and authenticated ftp sessions. (Currently there is no system-wide client config file. That way, trust must be assigned to individual users and not to whole systems.)

1. Connect from your client to the server setup in 6.8.1, "Setting up ftpd to use TLS" on page 365 and transfer the root certificate to the client (Example 6-14).

Example 6-14   Initial transfer of self-signed root CA certificate

```
# ftp -s nimrod
Connected to nimrod.
220 nimrod FTP server (Version 4.2 Mon Apr 9 11:38:07 CDT 2007) ready.
234 Using authentication type TLSv1
TLS Auth Entered.
TLS handshake succeeded, though Server signed it's own cert!
Certificate:
    Data:
        Version: 1 (0x0)
        Serial Number:
            ca:6d:1c:d0:7b:8d:ad:ea
        Issuer: C=US, ST=TX, L=Austin, O=IBM, OU=CA
        Validity
            Not Before: Apr 27 02:09:17 2007 GMT
            Not After : Apr 24 02:09:17 2017 GMT
        Subject: C=US, ST=TX, L=Austin, O=IBM, OU=Server
TLSv1/SSLv3 ( DHE-RSA-AES256-SHA ), 256 bits
Name (nimrod:root):
331 Password required for root.
Password:
230-Last unsuccessful login: Thu Apr 26 12:31:41 CDT 2007 on ftp from
brazos
230-Last login: Thu Apr 26 21:00:30 CDT 2007 on /dev/pts/1 from
sig-9-145-145-117.de.ibm.com
230 User root logged in.
200 PBSZ=0
200 Protection level set to Private.
ftp> cd /root/.tls/rootCA
250 CWD command successful.
ftp> get rootcert.pem
200 PORT command successful.
150 Opening data connection for rootcert.pem (1111 bytes).
TLSv1/SSLv3 ( DHE-RSA-AES256-SHA ), 256 bits
226 Transfer complete.
1130 bytes received in 0.1793 seconds (6.156 Kbytes/s)
local: rootcert.pem remote: rootcert.pem
ftp> by
221 Goodbye.
```

2. Adjust the user's .ftpcnt file to point to the CA certificate by only changing the one line and leaving all other options in that file as comments:

   ```
   CA_PATH          ./rootcert.pem
   ```

3. When you connect to your server, no further warning about (potentially bogus) self-signed certificates is displayed. The chain of trust is complete and the ftp session will start with something similar to Example 6-15.

Example 6-15   Secure FTP session using a complete chain of trust

```
# ftp -s nimrod
Connected to nimrod.
220 nimrod FTP server (Version 4.2 Mon Apr 9 11:38:07 CDT 2007) ready.
234 Using authentication type TLSv1
TLS Auth Entered.
Certificate:
    Data:
        Version: 1 (0x0)
        Serial Number:
            ca:6d:1c:d0:7b:8d:ad:ea
        Issuer: C=US, ST=TX, L=Austin, O=IBM, OU=CA
        Validity
            Not Before: Apr 27 02:09:17 2007 GMT
            Not After : Apr 24 02:09:17 2017 GMT
        Subject: C=US, ST=TX, L=Austin, O=IBM, OU=Server
TLSv1/SSLv3 ( DHE-RSA-AES256-SHA ), 256 bits
Name (nimrod:root):
```

When using commercially signed certificates, it is exactly the same, that is, the client has to know about the root CA's certificates (their signed public keys). As mentioned earlier in "Using commercially signed certificates" on page 370, those certificates can be downloaded from the Internet at http://www.columbia.edu/~ariel/good-certs/. They need to be installed on any client system. Any user's ~./.ftpcnf file needs to have them referred to (as shown in step 2 of the previous example). A secure ftp session will then look something like Example 6-16 (which is in fact the same as in Example 6-15 on page 373, except for the CA information):

Example 6-16   Same as previous example but using a third-party signed certificate

```
# ftp -s brazsbd
Connected to brazsbd.
220 brazsbd FTP server (Version 4.2 Mon Apr 16 13:51:54 CDT 2007)
ready.
234 Using authentication type TLSv1
TLS Auth Entered.
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 1000096493 (0x3b9c42ed)
        Issuer: C=US, O=Entrust, OU=Entrust PKI Demonstration
Certificates
        Validity
            Not Before: Apr 27 18:20:23 2007 GMT
            Not After : Jun 27 18:50:23 2007 GMT
        Subject: C=US, O=Entrust, OU=Entrust PKI Demonstration
Certificates, OU=Entrust/Web Connector, OU=No Liability as per
http://freecerts.entrust.com/license.htm, CN=www.foo.com
TLSv1/SSLv3 ( DHE-RSA-AES256-SHA ), 256 bits
Name (brazsbd:root):
```

Part 2

Appendixes

375

A

Crypto Lib in C (CLiC)

AIX operating system relies on CliC library in order to use such features as EFS, NFS Version 4, and Trusted Execution.

CliC library also provides application programmers with the cryptographic primitives required for applications that contain cryptographic features.

The level of abstraction of this library makes this library usable also for application developers who are not necessarily cryptography specialists.

API documentation is available as an easy to use HTML package.

Usage of this library offers various advantages that include:

► The IBM patented true random number generator is used to provide a key-generation process with high quality random numbers.

► Support for RSA signature generation and verification in hardware using the IBM Common Cryptographic Architecture (CCA) API.

► CliC is highly flexible and allows customization to include precisely the functions needed by your application, especially when it comes to embedded software on small devices.

377

- ► CliC is available for a a wide variety of platforms and architectures including:
  - – Windows 95/98/Me/NT/2000/XP/Vista
  - – Linux
  - – AIX
  - – OS/390® (MVS™ and z/OS®)
  - – OS/400® and MacOS
- ► High Level features include all necessarily features for supporting PKI Certificate management. Encryption and checksum support for Kerberos 5 is also provided.

CLiCToken and PKCS #11 Software Token Support

CLiCToken is a software crypto module derived from the RSA Security Inc. PKCS #11 Cryptographic Token Interface (Cryptoki). All cryptographic and PKI operations are carried out using the CLiC crypto libraries and persistent cryptographic objects will be stored in password-protected PKCS #12 files. CLiCToken is available as dynamic link libraries pkcs11-SW.so, pkcs11-SW.so64, and libpkcs11-SW.a for UNIX platforms.

The features of the PKCS #11 interface to CliC includes:

- ► PKCS #11 standard programming API
- ► PKCS #11 V2.20 compliance
- ► High cryptographic performance using CLiC V4 libraries
- ► Secure object storage in password-protected PKCS #12 files
- ► Support for multi-threaded access
- ► Support for multiple slots/tokens (PKCS #12 files)

Supported PKCS #11 mechanisms are described in Table A-1 on page 379.

Table A-1   PKCS #11 supported mechanisms

Mechanism	Encryption+ Decryption	Sign+ Verify	Digest	Key- gen	Wrap+ Unwrap	Min	Max
CKM_MD5			x			n/a	n/a
CKM_SHA_1			x			n/a	n/a
CKM_SHA256			x			n/a	n/a
CKM_SHA384			x			n/a	n/a
CKM_SHA512			x			n/a	n/a
CKM_RSA_PKCS_KEY_P AIR_GEN				x		512	4096
CKM_RSA_X_509	x	x				512	4096
CKM_RSA_PKCS	x	x			x	512	4096
CKM_MD5_RSA_PKCS		x				512	4096
CKM_SHA1_RSA_PKCS		x				512	4096
CKM_SHA256_RSA_PKC S		x				512	4096
CKM_SHA384_RSA_PKC S		x				512	4096
CKM_SHA512_RSA_PKC S		x				512	4096
CKM_MD5_HMAC		x				128	512
CKM_MD5_HMAC_GENE RAL		x				128	512
CKM_SHA1_HMAC		x				160	512
CKM_SHA1_HMAC_GEN ERAL		x				160	512
CKM_SHA256_HMAC		x				256	512
CKM_SHA256_HMAC_GE NERAL		x				256	512
CKM_SHA384_HMAC		x				384	512
CKM_SHA384_HMAC_GE NERAL		x				384	512

Mechanism	Encryption+ Decryption	Sign+ Verify	Digest	Key-gen	Wrap+ Unwrap	Min	Max
CKM_SHA512_HMAC		x				512	1024
CKM_SHA512_HMAC_GENERAL		x				512	11024
CKM_AES_ECB	x					128	256
CKM_AES_CBC	x					128	256
CKM_AES_CBC_PAD	x					128	256
CKM_DES_ECB	x					56	56
CKM_DES_CBC	x					56	56
CKM_DES_CBC_PAD	x					56	56
CKM_DES3_ECB	x					168	168
CKM_DES3_CBC	x					168	168
CKM_DES3_CBC_PAD	x					168	168
CKM_GENERIC_SECRET_KEY_GEN				x		128	1024
CKM_DES_KEY_GEN				x		8	8
CKM_DES2_KEY_GEN				x		16	16
CKM_DES3_KEY_GEN				x		24	24
CKM_AES_KEY_GEN				x		16	32

LDIF file for supporting AIX Security Expert

This appendix contains an example LDIF file for extending the schema to support AIX Security Expert, as referenced in 6.4.1, "LDAP server preparation" on page 346.

AIX Security Expert LDIF file

Example B-1 shows the AIX Security Expert LDIF file.

Example: B-1   AIX Security Expert LDIF

```
# File generated at 19:37:39 on 02/02/2007 from IBM LDAP schema version
1.5
# Module Name: AIXpert (v 1) ( AIXpert-oid )
# Dependencies:
#
dn:cn=schema
changetype: modify
add: attributetypes
attributetypes: (
  ibm-aixpertLabel-oid
  NAME 'ibm-aixpertLabel'
  DESC 'An unique lable of XML file'
  EQUALITY caseExactMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.15
  SINGLE-VALUE
  USAGE userApplications
  )
ibmattributetypes: (
  ibm-aixpertLabel-oid
  DBNAME( 'aixpertLabel'  'aixpertLabel' )
  ACCESS-CLASS normal
  LENGTH 256
  )

dn:cn=schema
changetype: modify
add: attributetypes
attributetypes: (
  ibm-aixpertXmlConfigFile-oid
  NAME 'ibm-aixpertXmlConfigFile'
  DESC 'AIXpert XML Configuration file'
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.5
  SINGLE-VALUE
  USAGE userApplications
  )
ibmattributetypes: (
  ibm-aixpertXmlConfigFile-oid
  DBNAME( 'aixpertXmlConfigF'  'aixpertXmlConfigF' )
  ACCESS-CLASS normal
```

```
    )

dn:cn=schema
changetype: modify
add: objectclasses
objectclasses: (
  ibm-aixAixpert-oid
  NAME 'ibm-aixAixpert'
  STRUCTURAL
  )

dn:cn=schema
changetype: modify
replace: objectclasses
objectclasses: (
  ibm-aixAixpert-oid
  NAME 'ibm-aixAixpert'
  DESC 'For storing AIXpert specific data'
  SUP top
  STRUCTURAL
  MUST ( ibm-aixpertLabel $ ibm-aixpertXmlConfigFile )
  )
```

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this book.

IBM Redbooks

For information about ordering these publications, see "How to get Redbooks" on page 386. Note that some of the documents referenced here may be available in softcopy only.

► *AIX 5L Differences Guide Version 5.3 Addendum*, SG24-7414
► *AIX 5L Differences Guide Version 5.3 Edition*, SG24-7463
► *AIX 5L Version 5.2 Security Supplement*, SG24-6066
► *Securing NFS in AIX An Introduction to NFS v4 in AIX 5L Version 5.3*, SG24-7204

Other publications

These publications are also relevant as further information sources:

► Garfinkel, et al., *Practical UNIX and Internet Security, Third Edition*, O'Reilly Media, 2003, 0596003234
► *Strengthening AIX Security: A System-Hardening Approach*, found at:

   http://www-03.ibm.com/servers/aix/whitepapers/aix_security.html

Online resources

These Web sites are also relevant as further information sources:

► Multilevel Security:

   http://en.wikipedia.org/wiki/Multilevel_security

How to get Redbooks

You can search for, view, or download Redbooks, Redpapers, Technotes, draft publications and Additional materials, as well as order hardcopy Redbooks, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Index

Symbols
/var/efs/users   62

Numerics
4764 accelerator   45
4764 Cryptographic Accelerator   49
4960 coprocessor   45
4963 accelerator   45

A
Access Control List   81
access key   136–138
accountability   276
accreditation range   313, 316
Active Directory   45
adding aixpert rules   352
Adding BFF files to the TSD   268
Adding non-BFF files to the TSD   269
Advanced Encryption Standard   34
AES   32, 34, 80
   default key length   62
   default mode   62
   file encryption   70
AES_256_CBC   38
AES-128_CBC   38
AIX Certifications   50
AIX Security
   Total AIX Capabilities   43
AIX Security Expert   14
   "Undo" Option   17
   Consistency Check   17
   overview   335
   Policy Distribution   18
AIX Security Expert LDIF file   346
AIX Subscription Service   56
aixpert   336
algorithm mode   72
ANSI   34
asymmetric algorithms   34
asymmetric encryption   32
Audit   10
audit header   297

Audit Policy   16
audit range   296
audit start   297
audit subsystem   296
   accountability   276
   MLS attributes   296
   MLS audit records   296
   MLS related event   297
auditpr   297
   E   297
   I   297
   S   297
   -s   297
   -v   297
auditselect   297
   -e   297
authorizations   27
   RBAC authorizations   174
   suggest syntax   202
   system defined   200
   user defined   200
   Using UID/GID authorization in an executable   168
availability   10, 276

B
Bell-LaPadula Model   41
Block Ciphers   35

C
CAPP   45, 51
CAPP profiles   51
CBC mode   36
Centralized AIX Security Expert Policy   18
Certifications   50
channels   313
chauth   204
chdev   179
chgroup   140
chmod   82–83
chrole   193
chuser   140, 314
cipher algorithm   72

IBM

Redbooks

AIX V6 Advanced Security Features Introduction and Configuration

(0.5" spine)
0.475"<->0.875"
250 <-> 459 pages

AIX V6 Advanced Security Features
Introduction and Configuration

Redbooks

A comprehensive overview of AIX V6 operating system security features

New features: Role Based Access Control (RBAC), Trusted AIX (Multilevel Security), and Trusted Execution

AIX V6 Security Expert enhancements

AIX Version 6.1 provides many significant new security technologies and security enhancements. The purpose of this IBM Redbooks publication is to highlight and explain the security features at the conceptual level, as well as provide practical examples of how they may be implemented. Some features are extensions of features made available in prior AIX releases, and some are new features introduced with AIX V6.

Major new security enhancements will be introduced with AIX V6 in 2007:

- Trusted AIX (Multilevel Security)
- Role Based Access Control (RBAC)
- Encrypted File System
- Trusted Execution
- AIX Security Expert Enhancements

This IBM Redbooks publication will provide a technical introduction to these new enhancements. The topics are both broad and very complex. This book will serve as an initial effort in describing all of the enhancements together in a single volume to the security/system hardening oriented audience.

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks