Tru64 UNIX

Security

Part Number: AA-RH95D-TE

June 2001

Product Version: Tru64 UNIX Version 5.1A or higher

This manual describes how to use, administer, and write programs for the Compaq Tru64 UNIX operating system with the optional enhanced security subsets installed. The Security Integration Architecture (SIA), DOP privileges, and access control lists (ACLs) are also documented in this manual.

Compaq Computer Corporation
Houston, Texas

Contents

3 Connecting to Other Systems

4 Common Desktop Environment

5 Using ACLs on Files and Directories

Part 2 Administrator's Guide to Security

6 Introduction for Administrators

7 Setting Up the Trusted System

10 Administering the Audit Subsystem

11 Administering ACLs

21 Programming with ACLs

Figures

Tables

About This Manual

This manual describes how to use, administer, and write programs for the Compaq Tru64™ UNIX operating system with the optional enhanced security subsets installed. It also provides information about traditional UNIX security and other optional security features.

Audience

Part 1 is directed toward general users. It is not intended for users of secure programs, because such programs typically hide the secure interface after the login has been completed.

Part 2 is directed toward experienced system administrators and is not appropriate for novice administrators. System administrators should be familiar with security concepts and procedures.

Part 3 is intended for programmers who are modifying or creating security-relevant programs (trusted programs) and anyone who modifies or adds to the trusted computing base. You should be familiar with programming in C on UNIX systems.

Organization

The manual is divided into three parts as follows:

Part 1: User's Guide to Security

This part describes the enhanced security features of the system that relate to the general user. It also includes general information about connecting to other systems and using a windows environment.

Part 2: Administrator's Guide to Security

This part explains concepts that are fundamental to administering a trusted operating system and describes tools and procedures for administrative tasks. It is both task-oriented and conceptual.

Part 3: Programmer's Guide to Security

This part describes the security features to those who modify or add security-relevant programs (trusted programs). It presents guidelines and practices for writing these programs and describes specific Tru64 UNIX interfaces. This part also describes the use of the security facilities: system calls, libraries, and databases.

This manual is organized as follows:

Part 1: User's Guide to Security

Chapter 1	Introduces the enhanced security features of the system from a user's point of view and defines the areas in which a trusted system expands the traditional UNIX system for security.
Chapter 2	Describes how to log in to the system and change passwords. It also discusses some common problems associated with passwords and logging in and how to avoid them.
Chapter 3	Discusses the security risks and security procedures for logging into remote systems. Protecting files from remote copies is also discussed.
Chapter 4	Discusses the Common Desktop Environment (CDE) features that enhance the security of a workstation. This chapter does not explain how to use CDE.
Chapter 5	Describes the ACL (access control list) features of the system and how users can most effectively use them.

Part 2: Administrator's Guide to Security

Chapter 6	Defines a trusted system and security concepts fundamental to system security. It also summarizes the trusted administrative roles, protected subsystems, and security databases.
Chapter 7	Describes how to set up the security databases and parameters for system operation and how to customize the system for your own site.
Chapter 8	Describes how to create and modify secure terminals.

Chapter 9	Describes how to use the Account Manager (`dxaccounts`) programs to create and maintain accounts. It also describes the authentication subsystem and centralized account management.
Chapter 10	Describes the audit subsystem and how it is configured and maintained. Summarizes audit record formats and presents guidelines for effective and high-performance audit administration. This chapter also summarizes the formats of the records written to the audit trail by the audit subsystem.
Chapter 11	Describes the installation and administration of the ACLs (access control lists) feature.
Chapter 12	Describes the operations that check for system and database integrity.
Chapter 13	Describes the Security Integration Architecture (SIA) and the associated `matrix.conf` files. The installation and deletion of layered security products is also discussed.
Chapter 14	Lists problems that can occur during system operation and suggests resolutions.

Part 3: Programmer's Guide to Security

Chapter 15	Describes the approach to examples used throughout this part and provides information about the trusted computing base.
Chapter 16	Provides specific techniques for designing trusted programs, such as whether the program is to be a directly executed command or a daemon.
Chapter 17	Describes the structure of the authentication database and the techniques for querying it.

Chapter 18	Presents the various user and group identities of the operating system and how you should use them, particularly the audit ID that is not a part of traditional UNIX systems. It also describes the contents of the enhanced (protected) password database.
Chapter 19	Presents guidelines for when trusted programs should make entries in the audit logs and the mechanisms for doing so.
Chapter 20	Documents the Security Integration Architecture (SIA) programming interface.
Chapter 21	Provides the programmer with the information needed to use ACLs (access control lists) in applications that run on Tru64 UNIX.
Appendix A	Lists the files provided in the system's trusted computing base (TCB).
Appendix B	Contains the default auditable events (`/etc/sec/audit_events`) and the default audit-event aliases (`/etc/sec/event_aliases`) files.
Appendix C	Provides the programmer with extended coding examples for trusted Tru64 UNIX systems.
Appendix D	Explains the naming convention used to keep Tru64 UNIX compliant with ANSI C.
Appendix E	Provides administrators with detailed security configuration procedures.
Appendix F	Provides security configuration procedures for systems in a cluster. Audit and NIS setup information is included.
Appendix G	Provides administrators with DOP privileges information.

Related Documentation

The following Compaq documents provide additional information about security issues in the Tru64 UNIX system:

Command and Shell User's Guide

Common Desktop Environment documentation

Installation Guide

System Administration

Programmer's Guide

Reference Pages

The following are documents available from O'Reilly and Associates, Inc. that will help you understand security concepts and procedures:

Computer Security Basics

Practical UNIX Security

UNIX: Its Use, Control, and Audit — Contact the Institute of Internal Auditors Research Foundation at 249 Maitland Avenue, Altamonte Springs, Florida 32701-4201.

The following are reference documents available from the United States Department of Defense that you may find useful:

Site Security Handbook (RFC 1244) — This handbook is the product of the Site Security Policy Handbook Working Group, a combined effort of the Security Area and User Services Area of the Internet Engineering Task Force. This RFC provides information for the Internet community and is available at `http://www.net.ohio-state.edu/hypertext/rfc1244/toc.html`.

Trusted Computer System Evaluation Criteria — U.S. Department of Defense, National Computer Security Center, DoD 5200.28-STD, December, 1985. This document, known as the *Orange Book*, is the U.S. Government's definitive guide to the development and evaluation of trusted computer systems. An online copy of the *Orange Book* is available at `http://nsi.org/Library/Compsec/orangebo.txt`

Password Management Guideline — U.S. Department of Defense, (CSC-STD-002-85), April 12, 1985. This document, known as the *Green Book*, supports the *Orange Book* by presenting a set of recommended practices for the design, implementation, and use of password-based user authentication mechanisms. An online copy of the *Green Book* is available at `http://www.radium.ncsc.mil/tpep/library/rainbow/CSC-STD-002-85.html`

A Guide to Understanding Audit in Trusted Systems — U.S. Department of Defense

The following document may be of interest to users outside the U.S.:

Information Technology Security Evaluation Criteria (ITSEC).

Icons on Tru64 UNIX Printed Manuals

The printed version of the Tru64 UNIX documentation uses letter icons on the spines of the manuals to help specific audiences quickly find the manuals that meet their needs. (You can order the printed documentation from Compaq.) The following list describes this convention:

G	Manuals for general users
S	Manuals for system and network administrators
P	Manuals for programmers
R	Manuals for reference page users

Some manuals in the documentation help meet the needs of several audiences. For example, the information in some system manuals is also used by programmers. Keep this in mind when searching for information on specific topics.

The *Documentation Overview* provides information on all of the manuals in the Tru64 UNIX documentation set.

Reader's Comments

Compaq welcomes any comments and suggestions you have on this and other Tru64 UNIX manuals.

You can send your comments in the following ways:

- Fax: 603-884-0120 Attn: UBPG Publications, ZKO3-3/Y32
- Internet electronic mail: `readers_comment@zk3.dec.com`

  A Reader's Comment form is located on your system in the following location:

  `/usr/doc/readers_comment.txt`

Please include the following information along with your comments:

- The full title of the manual and the order number. (The order number appears on the title page of printed and PDF versions of a manual.)
- The section numbers and page numbers of the information on which you are commenting.

- The version of Tru64 UNIX that you are using.
- If known, the type of processor that is running the Tru64 UNIX software.

The Tru64 UNIX Publications group cannot respond to system problems or technical support inquiries. Please address technical questions to your local system vendor or to the appropriate Compaq technical support office. Information provided with the software media explains how to send problem reports to Compaq.

Conventions

This document uses the following typographic conventions:

`%` `$`	A percent sign represents the C shell system prompt. A dollar sign represents the system prompt for the Bourne, Korn, and POSIX shells.
`#`	A number sign represents the superuser prompt.
`% cat`	Boldface type in interactive examples indicates typed user input.
file	Italic (slanted) type indicates variable values, placeholders, and function argument names.
`[ | ]` `{ | }`	In syntax definitions, brackets indicate items that are optional and braces indicate items that are required. Vertical bars separating items inside brackets or braces indicate that you choose one item from among those listed.
`...`	In syntax definitions, a horizontal ellipsis indicates that the preceding item can be repeated one or more times.
`cat(1)`	A cross-reference to a reference page includes the appropriate section number in parentheses. For example, `cat(1)` indicates that you can find information on the `cat` command in Section 1 of the reference pages.

Return	In an example, a key name enclosed in a box indicates that you press that key.
Ctrl/*x*	This symbol indicates that you hold down the first named key while pressing the key or mouse button that follows the slash. In examples, this key combination is enclosed in a box (for example, Ctrl/C).
Alt *x*	Multiple key or mouse button names separated by spaces indicate that you press and release each in sequence. In examples, each key in the sequence is enclosed in a box (for example, Alt Q).

Part 1

User's Guide to Security

1

Introduction for Users

The Tru64 UNIX operating system is delivered with an enhanced security optional subset and other optional security features. When the enhanced security subset subset is installed and configured, the system is referred to as a trusted system. The enhanced security features result in a system that can be configured to meet the C2 class of trust, as defined by the *Trusted Computer System Evaluation Criteria* (TCSEC, also called the *Orange Book*). The system also meets the F-C2 functional class as defined in the *Information Technology Security Evaluation Criteria* (ITSEC).

Although many of the requirements for maintaining the security of the trusted Tru64 UNIX system are the responsibility of your site's administrative staff, you have a responsibility, as a user of the system, to help enforce the security provided by the system. This chapter explains system capabilities and user responsibilities.

1.1 Security Features

The Tru64 UNIX system without the enhanced security subset installed provides traditional UNIX security, as described in the Tru64 UNIX manuals. Traditional UNIX security at the user level consists of basic login identification, authentication (password checking) and file permissions (discretionary access controls (DAC)). The following sections describe how enhanced security and the other optional security features extend traditional security.

The presence of the protected password daemon (`/usr/bin/prpasswdd`) indicates that enhanced security is enabled. To determine which of the security features are running on your system, see your system administrator.

1.1.1 Login Control Enhancements

Enhanced security features for login control may include the following:

- Recording of the last terminal used for a successful login
- Recording of the time of the last successful login
- Recording of the time of the last unsuccessful login attempt
- Recording of the number of consecutive unsuccessful login attempts
- Recording of the terminal used for the last unsuccessful login attempt

- Automatic account lockout after a specified number of consecutive bad access attempts
- A per-terminal setting for the delay between consecutive login attempts, and the maximum amount of time each attempt is allowed to complete the login before being declared a failed attempt
- A per-terminal setting for the maximum consecutive failed login attempts before locking any new accesses from that terminal
- Display information about last successful and last unsuccessful login attempts at login time.

1.1.2 Password Enhancements

Enhanced security provides the following features for password control:

- Configurable maximum password length, up to 80 characters
- Configurable password lifetimes
- Variable minimum password length
- System-generated passwords that take the form of a pronounceable password made up of meaningless syllables, an unpronounceable password made up of random characters from the character set, or an unpronounceable password made up of random letters from the alphabet (all letters are from ASCII)
- Per-user password generation flags, which include the ability to require a user to have a system-generated password
- Record of who (besides the user) last changed the user's password
- Password usage history

1.1.3 Audit Subsystem

One of the most useful security features of a Tru64 UNIX system is the audit subsystem, which an administrator can use to hold users accountable for their actions. The audit subsytem can record every relevant security event that happens on the system (for example, each file open, file creation, login, and print job submitted).

Each action is also stamped with an immutable audit ID (AUID) of the user who logged on, which allows all actions to be traced directly to a user. Users, by request to the system administrator, can use the audit trail to help re-create past events that affect the security of their accounts and data.

Users have no direct interaction with with the audit subsystem. The audit feature is discussed in detail in Chapter 10.

Audit is a kernel option and is available without the enhanced security subsets installed.

1.1.4 ACLs

Users on a Tru64 UNIX system can provide access granularity on files and directories down to a single user by using the optional Access Control List (ACL). An ACL can be associated with any file or directory on systems with file systems that support property lists. An ACL allows users to specify exactly how they want their files protected. See Chapter 5 for information on using ACLs.

1.2 User Accountability

A trusted system holds all users accountable for the actions that they perform on the system. When you log in, the system associates an audit ID (AUID) with your processes; the AUID remains stamped on processes regardless of the program being run. Even if you change your real or effective user ID (for example, by using `su` to become root or another user), the system still knows which authenticated user caused a specific action based on the identity recorded in the indelible AUID.

The system maintains an extensive authentication profile describing the characteristics and capabilities of each user – for example, the particular login restrictions on the user.

It is extremely difficult for an unauthorized user to break into a trusted system because of the extra security features added to the login procedure. In addition, in a trusted system you can more easily detect a penetration or attempted penetration into your account. Note, however, that these additional assurances are useless if you do not protect your password.

1.3 User Responsibilities

As a user of a trusted system, you must help protect the information that is stored and processed on the system. Specifically, you must do the following:

- Guard your password to protect against unaccountable access to your account.
- Apply strict discretionary access controls, including the use of access control lists, to protect your data from disclosure or destruction.
- Report all suspect activity to the system administrator, so that past events can be analyzed through the audit trail.

A trusted Tru64 UNIX system provides tools and mechanisms that help the system maintain the level of trust for which the system was designed. These are described in subsequent chapters.

2

Getting Started

This chapter explains how to log in to the system and use password facilities. Identification and Authentication (I and A) is the security term for all system procedures affecting logging in, changing passwords, and logging out. These procedures have been modified extensively in the trusted system, but these changes do not dramatically affect the way in which users perform their work on the system.

You should become familiar with the security functions and features of trusted Tru64 UNIX so you can learn to recognize any attempted (or successful) unauthorized use of your individual account or to the system in general.

2.1 Logging In

The login procedure on a system running under trusted Tru64 UNIX is similar to the procedure for nontrusted systems. This section describes the general process. See the `login`(1) reference page for details.

On a trusted system, you are occasionally required to change your password by using the `passwd` program (see Section 2.2.3 for a description of the circumstances). If you try to log in when your password needs to be changed, the `login` program calls the `passwd` program as part of the login procedure. You can also call `passwd` directly while you are logged in, as you can on a nontrusted system. Section 2.2 and the `passwd`(1) reference page describe the process.

The following example is a typical login on a trusted system:

```
login: juanita
Password: <nonechoed password>
```

The system then displays the date and time of the last successful and unsuccessful login:

```
Last successful login for juanita: date and time on tty03
Last unsuccessful login for juanita: date and time on tty03
```

Always check the successful and unsuccessful login information against your activity on the system. Any discrepancy means that someone has attempted to log in to your account (or did log in to your account). Report this activity

immediately to your system administrator or information system security officer (ISSO).

If your password is about to expire, the system displays a warning:

```
Your password will expire on date and time
```

The system administrator sets the warning interval on your system.

2.1.1 Authentication Profile

After a successful login, the system assigns the following attributes to your login shell:

- Login user ID (AUID, sometimes call the audit ID)
- Effective and real user IDs (EUID, RUID)
- Effective and real group IDs (EGID, RGID)
- Supplementary groups
- User audit control and disposition masks

As you log in, the system stamps your login process with an AUID. The AUID identifies you in the system auditing records so that you can be held accountable for your actions, as described in Section 1.1.3. The audit masks are used to calculate user-specific audit record collection, as set in your authentication profile. The other process identities serve the same purpose as in nontrusted systems.

2.1.2 Other Login Restrictions

An authorized user list can be created for a particular terminal. If such a list exists, your user name must appear in the list or you cannot log in at that terminal. In this case, the system displays the following message:

```
Not authorized for terminal access--see System Administrator
```

After a specified number of failed login attempts, the terminal can be disabled. This security precaution protects the system against break-in attempts by limiting the number of times someone can try to log in from a given terminal.

A terminal can also be explicitly locked by the system administrator. If the terminal is disabled or locked, the system displays the following message:

```
Terminal is disabled -- see Account Administrator
```

Your account can be disabled after a specified number of failed login attempts. Like disabling a terminal, this security precaution protects the system by limiting the number of times someone can try to guess your

password. Your account is also disabled automatically if your password exceeds its lifetime.

Your account can also be explicitly locked by the system administrator. If your account is disabled, the system displays the following message:

```
Account is disabled -- see Account Administrator
```

If any of these messages appear when you try to log in, report the occurrence to your administrative staff. If the terminal or your account has been disabled, the system administrator has to enable it again before you can log in.

2.2 Setting Your Password

A trusted Tru64 UNIX system differs from a nontrusted system in the way in which it generates and controls passwords. A number of options can be selected to determine how passwords are created, issued, changed, and revoked. These options control the following items and are discussed in detail in later sections:

- Whether you can change your password under any circumstances.
- Whether you have previously used a specific password.
- Whether you can choose your own password. (Section 2.2.1.)
- What type of password the system generates for you if you cannot choose your own. (Section 2.2.2.)
- When you are allowed to change your password and when you must change your password. (Section 2.2.3.)

In the trusted system, as in the untrusted system, the `passwd` command changes passwords. The prompts this command displays and your interaction with it, however, are different in the trusted system.

If you are not allowed to change your password and you try to run `passwd`, the system displays the following message:

```
Password request denied.
Reason: you do not have any password changing options.
```

In this case, you must contact your system administrator and arrange to have your password changed.

2.2.1 Choosing Your Own Password

If you are allowed to change your password, your account can be set up to allow you to select your password or to have the system generate one. These options determine the dialog the system starts when you invoke `passwd`. First, the system prompts you for your current password:

```
Old password:
```

Type in your old password. If you type it correctly, the system displays password change times:

```
Last successful password change for user: date and time

Last unsuccessful password change for user: date and time
```

Always check these dates and times. Although you might not remember exactly when you last changed your password, you should at least be able to decide if the times are reasonable.

The system administrator can allow you to choose one or more of the following password types for your account:

- System-generated random pronounceable syllables
- System-generated random characters, including punctuation marks and digits
- System-generated random letters
- Your own choice

The following example shows the prompt when all possible options are allowed:

```
Do you want (choose one option only):
     1  pronounceable passwords generated for you
     2  a string of characters generated for you
     3  a string of letters generated for you
     4  to pick your password


Select ONE item by number:
```

If you choose to pick your own password, the system prompts for the new password twice to avoid mistypings.

2.2.2 Choosing a System-Generated Password

The following example shows the dialog for a system-generated pronounceable password:

```
Generating random pronounceable password for user.

The password, along with the hyphenated version, is shown.

Hit <RETURN> or <ENTER> until you like the choice.

When you have chosen the password you want, type it in.

Note: Type your interupt character or "quit" to abort at any time.
```

```
Password: saglemot     Hyphenation: sag-le-mot

Enter password:
```

The hyphenated version is shown to help you pronounce the password so you can remember it more easily. You do not enter the hyphens. If you do not like the first password, press Return to see another one. When the system generates one that you want, enter it.

If you decide not to change your password, you can enter `quit` or use your interrupt character (typically Ctrl/C). The system displays the following message:

```
Password cannot be changed.  Reason: user stopped program.
```

The system also updates your last unsuccessful password change time.

The dialogue when you select one of the other system-generated password types is similar.

2.2.3 Understanding Password Aging

The system enforces a minimum change time, expiration time, and lifetime for each password. Passwords cannot be changed until the minimum change time has passed. This prevents you from changing your password and then immediately changing it back so that you do not have to learn a new password. If you try to change your password too soon, the system responds with the following message:

```
Password cannot be changed.
Reason: minimum time between changes has not elapsed.
```

A password is valid until its expiration time is reached. Once a password has expired, you must change that password before the system allows you to log in again. You will usually see a message at login time if your password is about to expire. You should change it when you see the message. If you are logged out when your password expires, you can change it as part of the login process when you next log in.

If the lifetime passes, the account is disabled. If you try to log in to a disabled account, the system displays an appropriate message. In this case, you must ask your system administrator to reenable your account, and you must change your password when you next log in.

2.3 Using the su Command

The `su` command allows you to work on the system temporarily under the user ID of another person. The `su` command starts a new shell process with

the effective and real user and group IDs of the other user. In the trusted Tru64 UNIX system, the AUID is not changed through an `su` transition. This means that all actions are accountable to the user who originally logged in to the system, regardless of the number of `su` transitions, even through root.

See the `su(1)` reference page for details.

2.4 Password Security Tips

The identification and authentication procedure described in the preceding sections is one of the most important security tools the system uses to guard against unauthorized access. Knowing a password and having physical access to a terminal or remote access through the network are all that an unauthorized user needs to gain access to a system.

Once such a user has logged on, he or she can steal data and corrupt the system in subtle ways. The amount of damage a penetrator can do increases as the account accessed has greater power on the system.

Remember, a penetrator's actions can be traced only to your account, and you will be held accountable. It is your responsibility to ensure that your account is not compromised.

Protect your password by following these guidelines:

- Never share your password. When you tell someone your password and let them log in to your account, the system loses its ability to hold individual users accountable for their own actions.
- Do not write down your password. Many system penetrations occur simply because a user wrote his or her password on a terminal. If a password must be recorded, keep it under lock and key.
- Never use an old password again. This increases the probability that someone can guess the password.
- Never type a password while someone is watching. It is possible to steal a password simply by watching someone type it. Be especially careful if you are using a workstation in a public area.
- If you are allowed to choose your own password, choose your password wisely:
  - Select passwords that are hard to guess.
  - Never use an ordinary word or a proper name, your spouse's, child's, or pet's name, your birthday, your address, or a machine name, even if these words are specified backward, permuted in some other way, or have a number added to the front or back.

– Always choose a password that contains some numbers or special characters. Always select different passwords for different machines, but never use the name of the machine, even permuted.

Your system administrator can set defaults for your site that perform automatic checks on passwords you specify.

Although these procedures add a small amount of effort to your login, they help to avoid system compromise.

2.5 Login and Logout Security Tips

In addition to following the password security tips, follow these login and logout guidelines:

- Check the system login and logout messages. When you log in, carefully check the reported last login and logout times to make sure they match what you remember as the last time you logged in and out. Make special note of login attempts during the time that you normally do not log in to the system. Report any discrepancies immediately to your system administrator so he or she can analyze the audit trail for the attempted penetration.
- Never leave your terminal unattended. Remember, someone who can run a program under your identity can cause great damage. It is much easier for a malicious user to take advantage of an unattended terminal than to coerce you into running a trojan horse program.
- Analyze unsuccessful login attempts. Note any login attempts where you thought you entered the correct password but the system reported it as incorrect, especially if you then log in successfully. If the time reported for the last unsuccessful login is not close to the current time, you might have typed your password into a login spoofing program, and someone may now know your password. Either change it immediately (if you are allowed to do so), or arrange with the system administrator to have it changed.

2.6 Problem Solving

The trusted Tru64 UNIX's mechanisms may be somewhat unfamiliar if you are accustomed to a nontrusted Tru64 UNIX system. If you are a new user, the extra complexity added to satisfy security requirements may create additional confusion.

The following sections provide a guide to common situations that cause users problems. Each description of a potential problem and its suggested solution should give you greater understanding of the security features that are exhibiting unexpected behavior.

2.6.1 Passwords

The trusted Tru64 UNIX system enforces two modes of password expiration:

- A password expires if its expiration time is reached. If your password expires, you must change it or arrange to have it changed (if the system administrator has not given you password change authorization) before logging into the system again. The system will not allow you to log in until your password is successfully changed.
- Your password dies if its lifetime is exceeded. In this case, your account is disabled; only the system administrator can reenable your account. You must change your password before using the system again after the system administrator reenables it.

Recall that the system warns you at login time that your password is about to expire. In this case, you should use the `passwd` command to change it before you log out. If your password expires while you are logged out, the `login` command calls `passwd` during the login process. See the `login`(1) and `passwd`(1) reference pages and Chapter 2.

The system also warns you if your password was changed by another user since you last logged in successfully. This message is to be expected if you cannot change your own password and the system administrator has changed it for you. If this message appears when you do not expect it, see your system administrator.

2.6.2 Background Jobs

If you are accessing the system from a character mode terminal, the `getty` command opens the `stdin`, `stdout`, and `stderr` file pointers to reference the terminal character device file. A program that manages to survive the user's logout can try to access the terminal because its file descriptors are retained. This is an open opportunity for login spoofing programs, because a background program can read the terminal file descriptor and be given some of the characters that are also requested by the `getty` and `login` programs for the new user session.

The Tru64 UNIX system invalidates all terminal file descriptors after logout. If a program tries to access the login terminal after logout, the access fails. One impact of this feature occurs when you are using `write` to communicate with another user, and that user logs out or the terminal is disconnected. The next message that you try to send causes `write` to exit with an error message, because it no longer has access to the other terminal.

Background jobs can be left running after you have logged out. If these jobs attempt to write to a terminal using the `write`() system call after logout, they receive a hangup signal, and the write fails. The behavior of the program depends on how it handles that error condition.

2.6.3 Sticky Directories

One of the UNIX permission bits is called the "sticky bit." In older UNIX systems, the sticky bit was set on executable files so that the system retained the program text in the swap area even after there were no active references to the program. This behavior was useful for some earlier computer architectures. On these early systems, the sticky bit for directories had no meaning.

Nontrusted Tru64 UNIX systems, trusted Tru64 UNIX systems, and some other recent UNIX variants use the sticky bit on directories to control a possible security hole.

Many commands use standard directories such as `/tmp` and `/var/tmp` to store temporary files. These directories are readable and writable by everyone so that all users can create and remove their own files in the temporary directories. Because the directories are writable, however, users can also remove other users' temporary files, regardless of the protection on the file itself.

Setting the sticky bit changes the semantics for writable directories. When the sticky bit is set, only the superuser or the owner of a process with the appropriate privilege can remove a file. Other users cannot remove files from such directories.

If you cannot remove a file from a directory to which you have discretionary write access, check the file's owner and the directory's sticky bit. The sticky bit is on if `ls` reports a `t` in the execute bit for others in a long listing. For example:

```
$ ls –ld /sticky

drwxrwxrwt  11 bin     bin      1904 Jan 24 21:56 /sticky
```

The administrator typically places the sticky bit on all public directories because these directories can be written by any user. These include the following directories:

- `/tmp`
- `/var/tmp`
- `/var/preserve`

Most systems combine the sticky directory approach with a policy of specifying restrictive `umask` values (for example, 077) for user accounts. In this case, temporary files are created as private files, which prevents users from altering or replacing files in shared directories. The user can determine only the file's name and attributes.

The trusted Tru64 UNIX system default `umask` is 077. If unauthorized users try to access such a file, they will be able to link the file from the temporary directory into a private directory, but will not be able to read the file even if a private copy can be saved.

Many systems create temporary directories as private file systems that do not allow links to user directory hierarchies.

2.6.4 SUID/SGID Clearing

Trusted Tru64 UNIX clears the following permission bits whenever it writes a file:

- Set user ID on execution (SUID)
- Set group ID on execution (SGID)

Be sure to restore these attributes when replacing a program.

2.6.5 Access Control Lists

An access control list is a mechanism that can be used to protect files. Although a file's owner/group/other permissions as shown by `ls` specify that a process has access to a file, the file's ACL may not allow the process access. This can be true even if the process has the same effective group as the group of the file.

In the following example, group `proj1` has write access to the file according to the `ls` display, but user `mario` in group `proj1` does not have write access according to the ACL. A process must pass all mandatory and discretionary checks before access to any object is allowed.

```
$ ls -l file-rw-rw-rw-   1 john     proj1        846
Jan 19 14:13 file

$ getacl file

# file:file
# owner:john
# group:proj1
user::rw-
group::rw-
user:mario:r--
group:dev:r--
other::rw-

$ date >file

file: Permission denied
```

Although the `ls` listing shows that the owning group has read and write access, the ACL shows that `mario` has only read access.

2.6.6 If You Cannot Log In

There are a number of reasons why a login attempt can fail on a trusted Tru64 UNIX system. The `login` program usually prints an informative message.

Mistyping the information required to log in is the most common reason for not being able to log in. When you do this, the system displays the following message and prompts you to enter your user name and your password:

```
Login incorrect
```

Try to log in again. The system limits the number of times you can enter an incorrect user name and password combination (see Section 2.1.2). If you exceed this limit, the system disables your account. If you forget your password, see your system administrator.

Most of the other reasons that you might not be able to log in are described in Section 2.1.2. The following list summarizes the reasons and explains what you should do:

- The terminal is disabled. See your system administrator, who must unlock the terminal before anyone can log in from it. If the terminal you normally log in from has been disabled, someone might have tried to break into the system from that terminal.
- Your name is not on the list of authorized users for the terminal. See your system administrator.
- Your account is disabled. See your system administrator to have your account reenabled. Your account might be disabled because you (or someone attempting to break in) have made too many unsuccessful login attempts. The account might also be locked by the system administrator.
- Your password has expired. See your system administrator to have your account reenabled. You can change your password during the next log in.

In general, you should see your administrator immediately if your account has been disabled or if anything unexpected happens when you try to log in.

3

Connecting to Other Systems

By connecting systems to each other, users have greater access to information; however, such connections also increase the security risks for each system. Responsible network security allows users some freedom, while protecting valuable files from unauthorized users.

Although the system administrator is responsible for most network security issues, individual users must be alert to security risks that affect their accounts and files.

The following networking protocols enable Tru64 UNIX users to communicate with other users on remote systems:

- Internet protocols (TCP/IP)
- Local Area Transport (LAT)
- The UUCP utility
- DECnet

Each protocol has its own scheme for handling communication between systems on a network. This chapter describes the security risks in using commands that connect to other systems using each of these protocols, and offers suggestions for minimizing those risks.

3.1 The TCP/IP Commands

The TCP/IP protocols are the most commonly used networking protocols running under Tru64 UNIX software. With TCP/IP, much of the network access to the computer is in the hands of users. The TCP/IP remote commands are described in the following sections.

3.1.1 The rlogin, rcp, and rsh Commands

The following commands enable you to communicate with remote systems:

`rlogin` — Lets you log in to a remote system. This command connects your terminal on the local host system to another login session either on a remote system or on the local host system. For more information, see the `rlogin`(1) reference page.

`rcp`	Lets you copy files to and from remote systems. For more information, see the `rcp`(1) reference page.
`rsh`	Lets you connect to a specified host and execute a command on the remote host. This command is a conduit to the remote command, passing it your input for processing and returning to you its output and any error messages that it generated. For more information, see the `rsh`(1) reference page.

A security risk in using the `rlogin`, `rcp`, and `rsh` commands lies in the network files `/etc/hosts.equiv` and `.rhosts`, which these commands check before connecting to a remote system.

3.1.2 The hosts.equiv File

The `/etc/hosts.equiv` file contains a list of host systems that are equivalent to your local host system. Users on equivalent hosts can log in to their accounts on the local host without typing a password. The user name on the remote and local host must be identical.

Equivalent hosts can be remote hosts or the local host. If the local host is listed in the `/etc/hosts.equiv` file, users logged in to the local host can remotely log in to their own accounts on the local host, without typing a password.

For security reasons, the `/etc/hosts.equiv` file does not allow a superuser logged in on a remote system to log in to the local host without typing a password.

Because the `/etc/hosts.equiv` file is a remote system's access key to your system, security-conscious system administrators leave this file empty or carefully restrict access to systems.

If the `/etc/hosts.equiv` file is empty, the only way a user on a remote host can log in to your account on the local host without typing a password is if the user's name is listed in your `.rhosts` file.

For more information, see the `hosts.equiv`(4) reference page.

3.1.3 The .rhosts File

The most common use of the `$HOME/.rhosts` file is to simplify remote logins between multiple accounts owned by the same user. If you have active accounts on more than one system, you may need to copy files from one account to the other or remotely log in to one account from the other. The `.rhosts` file is ideally suited to this type of use.

The $HOME/.rhosts file is a list of equivalent hosts that users can create in their home directories. This file is the user counterpart of the /etc/hosts.equiv file, although it has a narrower focus than its systemwide counterpart. The /etc/hosts.equiv file can affect the accounts of many users on a system. The .rhosts file affects only the individual user's account.

Your .rhosts file also enables users with your user name on equivalent hosts to log in to your account on the local host, without typing a password. Users must have a .rhosts file in their home directory.

Note

Equivalent hosts can be remote hosts or the local host. If the local host is listed in your .rhosts file, users with your user name, logged in to the local host, can remotely log in to your account on the local host, without typing a password. Including the local host in your .rhosts file enables you to remotely log in to your account and start a new session on the local host.

If you list another user's name next to the host name in your .rhosts file, that user can log in to your account on the local host; the remote user does not need an account on the local host or a .rhosts file in his or her home directory on the remote host. For example, the following entry in Peter's .rhosts file allows Paul to log in from rook as Peter without typing a password:

```
rook   paul
```

Your .rhosts file can expand the access that the /etc/hosts.equiv file grants to your account, but it cannot restrict that access. When a user executes the rlogin, rcp, or rsh command, that user's .rhosts file is appended to the /etc/hosts.equiv file for permission checking. The entries in the combined files are checked in sequence, one entry at a time. When the system finds an entry that grants access to the user, it stops looking. The entries in the /etc/hosts.equiv file are checked before the entries in the .rhosts file are checked. However, when the user is root, only the .rhosts file is checked.

If your security administrator excludes a host from the /etc/hosts.equiv file, then all users on that host are excluded. If you include that host in your .rhosts file, then users on that host are considered trusted and can log in to your account without entering a password. The converse is not true. If your system administrator includes a host in the /etc/hosts.equiv file, you cannot exclude users on that host from accessing your account. If you put a remote host and a user in the /etc/hosts.equiv file, that user on the remote host has access to all nonroot accounts on your host.

3.1.4 The ftp Command

The ftp command enables you to transfer files to and from a remote host, using the Internet standard File Transfer Protocol. In autologin mode, ftp checks the .netrc file in your home directory for an entry describing an account on the remote host. If no entry exists, ftp uses your login name on the local host as your user name on the remote host, and prompts for a password and, optionally, an account for login. Because your ftp login to a remote system is in essence a remote login to that system, you have the same access to files as if you, rather than ftp, had actually logged in. For more information, see the ftp(1) reference page.

A security risk in using ftp is the practice of creating the anonymous account, a generic account that the ftp command recognizes. The anonymous account usually has a commonly known password or no password, and it allows users to log in and transfer files to or from your system from a remote system with no audit trail. System administrators concerned with network security often avoid creating such anonymous accounts or carefully restrict which files can be copied or written.

You should know and follow the security policy on using ftp for file transfers to remote systems. Talk to your system administrator about the security controls at your system.

3.1.5 The tftp Command

The tftp command provides an interface to the Internet standard Trivial File Transfer Protocol. Like the ftp command, this command enables you to transfer files to and from a remote network site. However, the tftp command does not request a password when you attempt to transfer files. Therefore, any user who can log in to a system on the network can access remote files with read and write permission for other. Because the tftp protocol does not validate user login information, setting proper permissions on your files is the only real protection from unauthorized access.

The tftp command is shipped on the system but is turned off by default. To protect your system, avoid using tftp, if possible, or limit the directories that tftp can access.

3.1.6 Remote Connection Security Tips

Follow these guidelines to protect your files against attack through the rlogin, rcp, and rsh commands:

- Check your file permissions. Your home directory should deny all access to other, and write access to group. The permissions on the command and configuration files, such as .profile, .login, .logout, .cshrc, and .forward, should deny all access to group and other. For example,

use the chmod command to change the protections on those files from your home directory, as follows:

```
$ chmod 750 $HOME
$ chmod 600 .profile .login .logout .cshrc .forward
```

If you do a long listing of your home directory, your file protections should look like these:

```
$ ls -al

drwxr-x---  9   fields     512  Jun 13 11:46 .
-rw-------  1   fields     419  Jun 2  08:28 .login
```

Use the chmod command to set the permissions on your .rhosts file to 600. The *Command and Shell User's Guide* discusses protecting your files and directories.

- Include in the .rhosts file only the current remote hosts from which you would like to issue remote commands. It is wise to list only hosts on which you have accounts. If you are unsure about which hosts to include in this file, check with your system administrator.
- You should be the owner of your .rhosts file, and it must not be a symbolic link to another file.

3.2 LAT Commands

Your system administrator can increase the security of the LAT (Local Area Transport) protocol service by configuring LAT groups of hosts that can communicate only with each other or through specified terminals. A host can be set up to listen for connections from certain groups of terminal servers, while ignoring connections to all other LAT servers. For more information on using the LAT protocol, see the latcp(8) reference page.

3.3 The UUCP Utility

The UUCP utility is a group of programs that enable you to connect to remote systems using a modem and telephone lines. The UUCP utility, which is available on most UNIX systems, enables you to transfer files between remote systems and the Tru64 UNIX operating system. In addition, your system can use UUCP to send and receive mail across telephone lines.

Several UUCP commands can present security concerns:

- uucp
- tip
- cu
- uux

3.3.1 The uucp Command

The uucp command is the main interface to the UUCP utility.

The UUCP utility enables users on remote systems to access those files and directories for which the system administrator has granted permission. The uucp command allows any user to execute any command and copy any file that is readable or writable by a UUCP login user. Individual sites should be aware of this potential security risk and apply any necessary protections.

Your system administrator exercises certain security measures when installing and setting up the UUCP utility. However, it is important for you to take the following actions to protect against unauthorized use of this powerful utility through the uucp command:

- Create a directory in your account for UUCP. Use only this directory for all UUCP transactions.
- Use the chmod command to set the sticky bit on the UUCP directory. When the sticky bit is set on a directory, only root or the owner of a file can remove files from the directory. While you are operating under UUCP, you will not be able to remove those files while the sticky bit is set, and you may have a disk space problem. If this happens, remove the sticky bit from your directory and remove the excess files. The following example sets the sticky bit on the documents directory:

  ```
  $ chmod 1777 documents
  ```

- Until you set up a separate UUCP directory, always copy files to or from the /usr/spool/uucppublic directory.

For more information on setting the sticky bit, see the chmod(1) reference page. For more information on the UUCP utility, see the uucp(1) reference page.

3.3.2 The tip and cu Commands

The tip and cu commands enable you to call another system, log in, and execute commands while you are still logged in to your original system. The tip and cu commands are two different interfaces to the same program. The cu program allows you to be logged in on both systems at the same time, executing commands on either one without dropping the communications link. The tip command connects you to a remote system and allows you to work on the remote system as if logged in directly. You need only tell tip or cu what telephone number to call.

The following example shows a session using the cu command:

```
$ cu 4783939
```

```
connected
login:
```

A security concern about using the `tip` and `cu` commands is that everything you type is read by the command and passed to the remote system. This can be dangerous if the remote system is not a trusted system. A trojan horse version of `cu`, for example, could store your login name and password on a remote system. Follow these general security guidelines for using commands that start remote sessions:

- Be sure that the program you are using is the authentic program. Do not use a terminal that seems already to be running `tip` or `cu`; reinvoke the command using the full path.
- Do not use an automatic login procedure, such as sending your remote password from a file on the local computer.
- If you are capturing the session transcript into a local file, begin the capture only after completing remote login. Capture only the data you need; avoid capturing the dialogue you used to obtain the data.
- Avoid leaving your terminal or using your terminal for other things while a remote session is in progress. If your connection with the remote system is broken, immediately reestablish contact. Using the `ps -e` command, check to see if your first session left any processes suspended and kill those processes with the `kill -9` command.

For more information, see the `tip`(1) and `cu`(1) reference pages.

3.3.3 The uux Command

The `uux` command runs a specified command on a specified system while enabling you to continue working on the local system. The command gathers various files from the designated systems, if necessary. It then runs a specified command on a designated system. Users can direct the output from the command to a specified file on the designated system. For security reasons, many installations permit `uux` to run only the `rmail` command.

See the `uux`(1) reference page for more information.

3.4 The dlogin, dls, and dcp Commands

If DECnet is installed on your system, you can use the following DECnet commands to communicate with remote systems running the DECnet protocol:

- `dlogin`
- `dls`

- `dcp`

Your system administrator can increase DECnet security on your system by not creating a generic guest account for remote DECnet connections. Without this default user account, remote users must specify a valid user name and password either on the command line or interactively. For example, to copy a file from one system to a remote UNIX system without a default user account, you would have to type the following command:

```
$ dcp localfile rem_node/rem_user::/rem_path/file

Password for rem_node/rem_user:: ?:
```

If you are connecting to a remote system that has no default user account, you should not include the password information in the command. If you do not specify a password, you will be prompted for one. This provides more security because some shells (for example, the C shell) can maintain a history file. If you keep a history file and enter your password in clear text on a command line, the password is stored in the history file.

4

Common Desktop Environment

This chapter discusses CDE (Common Desktop Environment) features that improve the security of a workstation.

4.1 External Access to Your Display

When you log in to a workstation and create a session, the CDE login program sets the initial controls on access to the workstation. Any client that has access to the workstation display has full access to all events and resources of the X server, including the following:

- The ability to capture events such as keyboard and mouse events that include passwords or confidential information, even if it is not echoed when typed.
- The ability to send simulated events, including keystrokes, to windows on the display. For instance, a malicious user could send synthetic keystrokes to a terminal emulator window forming commands that would be executed under the UID of the logged in user.
- The ability to capture a snapshot of any part of the screen by setting the background pixmap of a window to None.
- The ability to display windows on the screen that masquerade as known programs (trojan horses).

4.2 Controlling Network Access to Your Workstation

Controlling access to your workstation display is the key to creating a secure workstation environment. Access is controlled by the following mechanisms:

Host Access

Any client on a host in the host access control list is allowed access to the X server. This is most useful in an environment where everyone trusts everyone, or where each host has only one user. The list of authorized hosts is controlled by the `xhost` command. The CDE login program by default starts the session with an empty host access list and uses the more secure access mechanisms. You can add hosts to the host access control list by putting `xhost` commands in your `.dtprofile`. See the `xhost`(1X) reference page for more information.

MIT-MAGIC-COOKIE-1

The CDE login program creates a 128-bit **cookie** when it starts the X server. A client is only allowed to connect if it presents the same cookie with the connection setup request. The cookie is chosen so that it is hard to guess, but it is transmitted on the network without encryption, so that it is susceptible to snooping. This is most useful where multiple users use the same machine, but network snooping is not an issue.

XDM-AUTHORIZATION-1

This is similar to MIT-MAGIC-COOKIE-1, except that the cookie data is encrypted using DES along with a time stamp to prevent snooping. This may not be available in all countries.

See the `XSecurity`(1X) reference page for more information about access control mechanisms.

Remember that hosts that are authorized to access your workstation display can read from it, write to it, and copy from it at any time. Restricting access is the only way to prevent users from taking a snapshot of the contents of your workstation display.

4.2.1 Host Access Control List

The X server maintains a host access control list to decide whether to allow connections from clients on a particular host. When a session is started, the host access control list is initialized from the file called `/etc/X`*n*`.hosts`, where *n* is the display number. The host access control list is reinitialized for each session even if the server is not restarted. For example, `X0.hosts` is the initial list of hosts that are authorized to connect to display 0, which is usually the default display.

Each line of the `/etc/X`*n*`.hosts` file is the name of a host, optionally preceded by the name of its address family. Adding a host to the host access control list enables any client running on that host to access the server.

The `xhost` utility can be used to add or remove hosts from the host access control list. See the `xhost`(1X) reference page for more details.

Allowing remote systems to access your account on a workstation is a security concern. Check with your security administrator before authorizing additional hosts to use your workstation display.

4.2.2 Authorization Data

For the MIT-MAGIC-COOKIE-1 and XDM-AUTHORIZATION-1 mechanisms, the data needed by the server to generate the authorization

information is the same as the data needed by the clients. The login program stores the authorization data in a file. The default is `$HOME/.Xauthority`.

This is particularly useful in an environment where the users' home directories are exported by NFS. Once you log in on a workstation, your authorization data is available to authorize connections from any host that has his home directory is mounted.

4.2.3 Using the X Authority File Utility

The `xauth` program allows you to run client applications on other hosts that do not share the home directory. You use the `xauth` program to edit and display the authorization information used to connect to the X server. You usually use this program to extract authorization records from one machine and merge them in on another (as is the case when using remote logins or granting access to other users). Note that this program does not contact the X server.

Using the X authority file utility is the recommended method of securing your workstation. For more information, see the `xauth`(1X) reference page and the *X Window System Environment* manual.

4.3 Protecting Screen Information

Any client given access to the display can access all the resources of the display, including all windows. If you display sensitive information on the screen, be careful about what hosts, if any, are enabled in the host access list. If a host is enabled, any user who can log in to that host has full access to the display resources.

By default, the `$HOME/.Xauthority` file is protected so that it is owned by the login user and is readable only by the login user. If you use an authorization mechanism that stores data in the `$HOME/.Xauthority` file and your home directory is exported with NFS, the workstation security is only as good as the NFS file security.

Normally when a window is created with a background pixmap resource set to the special value of None, the initial contents of the window are set to whatever is on the screen within the bounds of the window when it is first mapped. To prevent other clients from capturing snapshots of the screen using this feature, there is an optional command line parameter to the X server that disables it. To set the parameter, add the `+sec_objectreuse` option to the X server command line in the `/var/dt/Xservers` file.

4.4 Blocking Keyboard and Mouse Information

By default, dtterm and xterm windows ignore synthetic keyboard and mouse events sent by other clients. This security feature prevents unauthorized users from sending potentially destructive commands to your workstation when it is idle.

The ability of a dtterm window to block information sent from another host is set by a resource called allowSendEvents. If it is necessary to allow all dtterm or xterm windows to accept synthetic events, the resource can be set in a $HOME/.Xdefaults file.

The following example shows a line in the .Xdefaults file that sets the allowSendEvents resource to true, allowing other clients to send keyboard or mouse information to any window that you create:

```
dtterm*allowSendEvents:  true
```

If it is only necessary to enable synthetic events for specific terminal windows for some specific purpose, it is better to set the allowSendEvents resource with the -xrm command line option. For example:

```
% xterm -xrm "*allowSendEvents: true"
```

Alternatively, the special terminal emulator can be started with a different resource name and the $HOME/.Xdefaults can set the resource only for terminal emulators run with the special name. For example, if the $HOME/.Xdefaults file contains this line:

```
foo*allowSendEvents: true
```

The synthetic events are allowed only by terminal emulators that are run as follows:

```
% /usr/dt/bin/dtterm -name foo
```

Other applications may or may not accept synthetic keyboard and mouse events.

4.5 Pausing Your Workstation

In a CDE environment, you can pause your current session. This locks your workstation without ending your session. Your screen is cleared, and the system displays the screen saver. You can resume your session any time without re-creating your screen environment.

To put your current session on hold, click on the Padlock on the dashboard. Your screen is cleared and the Continue Session box is displayed. To continue your session, type your password then and press Return. Once your password is verified, your session resumes.

CDE provides a Screen Lock feature that automatically pauses your session after a period of inactivity. The Screen Lock feature is disabled by default and works with or without enhanced security enabled. The Screen Lock Start Lock (inactivity) time period can be set from 1 to 120 minutes with the default being 30 minutes.

Compaq recommends that you enable the Screen Lock feature and set the Start Lock time at a maximum of 10 minutes.

4.6 Workstation Physical Security

Workstations present security problems because they are typically found in ordinary offices, rather than the more easily protected environment of the computer room.

It is possible for someone who gains access to a workstation, to get superuser status on that system, and consequently on other systems. One method is to boot the system into single user mode.

If your office has a locking door, lock the door when you are away from your system.

You must also protect your removable media, such as tape cartridges and floppy disks by locking up all floppy disks and tape cartridges when they are not in use.

Some workstations allow a console password to be set. When a console password is in use, only a default boot can be done without a password. Check your hardware and firmware documentation for more information about console passwords.

5

Using ACLs on Files and Directories

This chapter describes the access control list (ACL) features for files and directories and explains how to use them effectively. It also describes the structure of ACLs and the methods used to create and maintain them.

The Tru64 UNIX ACLs are based on the POSIX P1003.6 Draft 13 standard with some Draft 15 extensions.

5.1 Traditional UNIX File Permissions

The traditional UNIX file permissions are displayed using the `ls` command with the `–l` flag. The permissions indicate what kind of access (that is, the ability to read, write, and execute) is granted to the owner and groups on your system. Traditional UNIX file protections allow some control over who can access your files and directories, but they do not allow you to define access for individual users and groups beyond the owning user and the owning group. The following is a brief review of UNIX file permissions.

Each file and each directory has nine permissions associated with it. Files and directories have the following three types of permissions:

- `r` (read)
- `w` (write)
- `x` (execute)

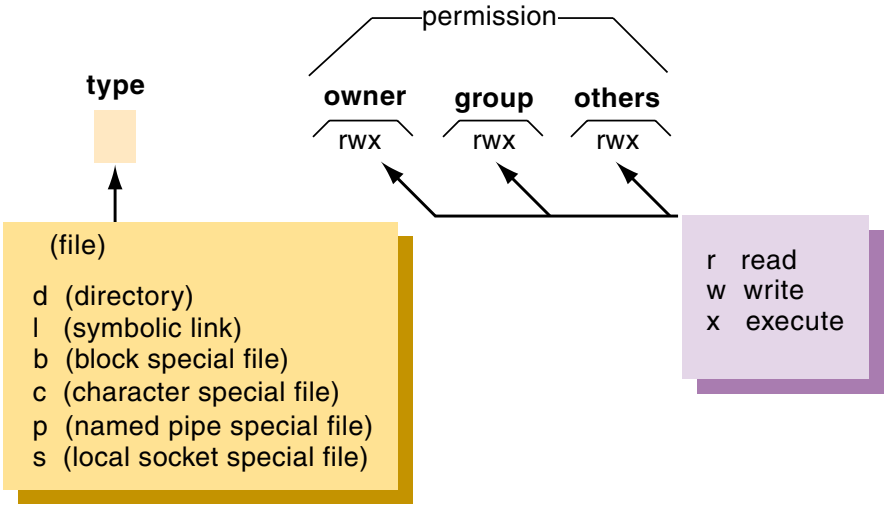
These three permissions occur for each of the following three classes of users:

- `u` (user/owner)
- `g` (group)
- `o` (all others; also known as world)

The `r` permission allows users to view or print the file. The `w` permission allows users to write to (modify) the file. The `x` permission allows users to execute (run) the file or to search directories.

Figure 5–1 illustrates the traditional permissions fields.

Figure 5–1: File and Directory Permission Fields



The user/owner of a file or directory is generally the person who created it. If you are the owner of a file, you can change the file permissions with the `chmod` command.

The `group` specifies the group to which the file belongs. If you are the owner of a file, you can change the group ID of the file with the `chgrp` command.

Note

If you do not own a file, you cannot change its permissions or group ID unless you have superuser authority.

The meanings of the three types of permissions differ slightly between ordinary files and directories. See Table 5–1 for more information.

Table 5–1: Differences Between File and Directory Permissions

Permission	For a File	For a Directory
r (read)	Contents can be viewed or printed.	Contents can be read, but not searched. Normally `r` and `x` are used together.
w (write)	Contents can be changed or deleted.	Entries can be added or removed.
x (execute)	File can be used as a program.	Directory can be searched.

See the *Command and Shell User's Guide* for a complete description of traditional UNIX file permission bits.

5.2 Why Use ACLs?

To allow you to specify access for individuals and groups, Tru64 UNIX files and directories can be configured with an optional attribute called the Access Control List (ACL). An ACL can be associated with any file or directory on systems with ACLs enabled and with file systems that support property lists. Contact your system administrator to find out if ACLs are enabled on your system and the file systems that you are using. See the `acl(4)` and `proplist(4)` reference pages for more detailed information.

Files only have a single ACL associated with them. A directory can have three ACLs associated with it. The access ACL is used similarly to the ACL on a file. But the default ACLs, if they exists, determine the ACLs created for descendents of the directory.

To allow maximum protection of files, an ACL extends the traditional protection scheme in three ways:

- With separate access control specifications for individual users and groups. Each entry in an ACL identifies an individual user or group and associates permissions with the user or group identified.
- By limiting the permissions that can be granted to individually specified users and groups.
- By allowing user and group permissions to be automatically specified on file creation by the use of default ACLs on directories. If directory hierarchies are maintained on a per-project basis, it can be useful to establish different access controls at the directory level. You can define default ACLs for a directory that are inherited by files and subdirectories in that directory when they are created.

5.3 ACL Status on Your System

The system administrator can enable and disable ACLs on your machine. When ACLs are enabled, the full functionality of ACLs is available and ACL access checking is enforced (where appropriate).

If ACLs are disabled, you can still set and retrieve ACLs on files and directories. However, ACLs are not validated or checked to determine access. The ACLs commands (`dxsetacl`, `setacl`, and `getacl`) display a warning message if ACLs are not enabled on your system.

Caution

Disabling ACLs on the system may allow processes access to files and directories to which ACLs disallow access. It is especially important that systems that share files using NFS, have a common security domain.

See your system administrator to determine if ACLs are enabled on your system.

5.4 Setting and Viewing ACLs

The following commands display and modify ACLs:

`dxsetacl`	A graphical interface that lists ands changes the ACLs on files and directories.
`setacl`	Changes the ACLs on files and directories.
`getacl`	Lists the ACLs on files and directories.

An ACL is viewed by using the `dxsetacl` GUI or the `getacl` command. The `dxsetacl` interface is found in the CDE Desktop Applications under the Applications Manager or you can open it from the command line as follows:

```
% /usr/bin/X11/dxsetacl &
```

An ACL is created and initialized when an object is created. You can change the ACLs on objects that you own by using the `setacl` command for files and directories. These commands take as an argument ACL entries that modify the ACL on the object.

If there is no access ACL associated with the file or directory, the standard UNIX permission bits are shown in the ACL format. If you are attempting to display a default ACL and there is no default ACL, an informational message is displayed.

These commands are used in examples later in this chapter. Refer to the `dxsetacl`(1),`setacl`(1), and `getacl`(1) reference pages for more detailed information. The `acl`(4) reference page also contains useful information about ACLs.

5.4.1 Using the dxsetacl Interface

The dxsetacl command is used to view and change ACL using a graphical format. The dxsetacl interface is found in the CDE Desktop Applications under the Applications Manager or you can open it from the command line as follows:

```
% /usr/bin/X11/dxsetacl &
```

See the dxsetacl(1) reference page and the online help for dxsetacl for more information.

5.4.2 Using the setacl Command

The setacl command is used to modify, add, and remove entries from existing ACLs. You can set the ACL of a file only if you own the file or you are superuser. See the setacl(1) reference page for more information.

5.4.3 Using the getacl Command

The getacl command lists the ACL on a file in a manner similar to the ls command. See the getacl(1) reference page for more information.

5.4.4 ACLs and the ls Command

The ls -l command displays the access allowed for the owning-user, the owning-group, and others on the file or directory. The ls -l command does not display the access allowed or denied by the access ACL (if any). To see the access allowed or denied by the access ACL, use the getacl command.

5.5 ACL Structure

An access control list consists of a number of ACL entries, each of which contains three fields, as follows:

- A keyword identifying the entry type
- A qualifier field that may contain a group or user ID or name
- A permission specification

The external (printable) representation of an ACL consists of comma (,) or newline-separated entries. The fields in the ACL entries are separated by colons (:). The following example shows typical ACL entries:

```
user::rwx
user:juanita:r-w
user:sam:r-x
group::rwx
other::---
```

The ACL entry keywords and qualifiers are defined as follows:

`user::` — A `user` entry with a NULL qualifier field defines the permissions of the user who owns the file. This entry (called an owning-user entry) is always identical to the `user` permission bits. An ACL must contain exactly one `user::` entry.

`user:` — A `user` entry with a non-NULL qualifier field defines the permissions of the user specified by the qualifier field. The qualifier field must contain either a username or a UID. An ACL may contain zero or more `user:` entries.

`group::` — A `group` entry with a NULL qualifier field defines the permissions of members of the group that owns the file. This entry (called an owning-group entry) is always identical to the group permission bits. An ACL must contain exactly one `group::` entry.

`group:` — A `group` entry with a non-NULL qualifier field defines the permissions of members of the group specified in the qualifier field. The qualifier field must contain either a groupname or a GID. An ACL may contain zero or more `group:` entries.

`other::` — The `other` entry is only valid with a NULL qualifier. This entry defines the permission of all users that did not match any of the other entries in the ACL. This entry is always identical to the `other` permission bits. An ACL must contain exactly one `other::` entry.

The characters in the permissions field are the same as the characters the `ls` command displays for the traditional permission bits and are in the same order: `r` for read access, `w` for write access, and `x` for execute or search access. When a hyphen (–) character is used in place of one of the other characters, it indicates denial of that access.

Table 5–2 illustrates and explains typical ACL entries.

Table 5–2: Example ACL Entries

Entry	Matching Criteria
`group:acct:r--`	Matches all users in group `acct` and grants read permission.
`user:joe:rw-`	Matches user `joe` and grants read and write permission.
`user::rwx`	Matches owner of object, even if owner changes after the file is created, and grants read, write, and execute permission.
`group::r--`	Matches owning group of object, even if owning group changes after the file is created, and grants read permission.
`other::r--`	Matches all users and all groups except the owning user and group and any other users and groups listed in ACL entries. Grants read permission.

5.6 Access Decision Process

When a process requests access to a file or directory, the following checks are made in the following order:

1. If the process has the superuser privilege, access to the file or directory is granted. The access ACL and the permission bits are not checked.
2. If ACLs are not enabled, or they are enabled and there is no access ACL associated with the file or directory, the traditional UNIX permission bit checks are used.
3. The access ACL for the file or directory is checked as follows:

    a. If the process is the owner of the object, the permissions in the owning `user::` entry are granted. Any other ACL entries are not checked. This is identical to using the `user` permission bits.

    b. If the UID of the process matches a UID listed in a `user:` entry or resolves to a username listed in a `user:` entry, the permissions in the entry are granted. Any remaining ACL entries are not checked.

    c. If the GID of the process matches the GID of the file, or if one of the supplementary groups of the process matches the GID of the file, the process is granted the union of the permissions of the `group:` entry and any matching `group:` entries as described in the next list item.

    d. If the GID of the process matches the GID of any `group:` entries, or resolves to a groupname listed in any `group:` entries or if the GID or groupname of any of the supplementary groups of the process match any `group:` entries of the ACL, the process is granted the

union of the protections of all matching group entries. For example, for a user belonging to group `sales` and group `eng`, if the access ACL on a file grants read access to group `sales` and write access to group `eng`, the user is granted read and write access to the file.

e. The permissions in the `other:` entry are granted. This is identical to using the `other` permission bits.

The default ACLs on a directory are used for file and directory creation. They are not used for access decisions.

Note

A file or directory with traditional UNIX permission bits and a file or directory with an access ACL containing only the three required entries (`user::`, `group::`, and `other::`) are indistinguishable.

5.7 ACL Inheritance

When a file or directory is created, it may inherit ACLs from its parent directory. A file only has one ACL associated with it, an access ACL. The access ACL determines access to the file as discussed in Section 5.6. A directory can have three ACLs associated with it: an access ACL, a default access ACL, and a default directory ACL.

5.7.1 Inheritance Matrix

The default ACLs determine what ACLs are inherited by files and subdirectories created in a parent directory, as follows:

- If a parent directory has no default ACLs:
  - A new file created in that directory is given:

ACL Type	Status
Access ACL	None

  - A new subdirectory created in that directory is given:

ACL Type	Status
Access ACL	None
Default access ACL	None
Default directory ACL	None

    The permission bits are set as with traditional UNIX.

- If a parent directory has a default access ACL, but no default directory ACL:
  - A new file created in that directory is given:

ACL Type	Status
Access ACL	Parent's default access ACL

  - A new subdirectory created in that directory is given:

ACL Type	Status
Access ACL	Parent's default access ACL
Default access ACL	Parent's default access ACL
Default directory ACL	None

- If a parent directory has no default access ACL, but does have a default directory ACL:
  - A new file created in that directory is given:

ACL Type	Status
Access ACL	None

  - A new subdirectory created in that directory is given:

ACL Type	Status
Access ACL	Parent's default directory ACL
Default access ACL	None
Default directory ACL	Parent's default directory ACL

- If a parent directory has both a default access ACL and a default directory ACL:
  - A new file created in that directory is given:

ACL Type	Status
Access ACL	Parent's default access ACL

  - A new subdirectory created in that directory is given:

ACL Type	Status
Access ACL	Parent's default directory ACL
Default access ACL	Parent's default access ACL
Default directory ACL	Parent's default directory ACL

Setting the default ACLs on a directory does not modify the ACLs on files and subdirectories that already exist in the directory.

5.7.2 ACL Inheritance Examples

Some examples of ACL inheritance follow:

- Assume that the directory `foo` contains no default ACLs, and the following command is issued to give `foo` a default access ACL:

  ```
  % setacl –d –u user::rw-,group::r--,other::r--,user:jdoe:rw–\
    foo
  ```

  Any file or directory that is created within the directory `foo` now inherits the following ACL as the access ACL:

  ```
  #
  # file: foo
  # owner: smith
  # group: system
  #
  user::rw-
  user:jdoe:rw-
  group::r--
  other::r--
  ```

- Assume that the directory `foo` contains no default ACLs, and the following command is issued to give `foo` a default directory ACL:

  ```
  % setacl –D –u user::rw-,group::r--,other::r--,\
    user:jdoe:rwx foo
  ```

  Any directory that is created within the directory `foo` now inherits the following ACL as the access ACL, as well as its default directory ACL:

  ```
  #
  # file: foo
  # owner: smith
  # group: system
  #
  user::rwx
  user:jdoe:rwx
  group::r--
  other::r--
  ```

- Assume that the directory `foo` contains no default ACLs, and the following commands are issued to give `foo` a default access ACL and a default directory ACL:

  ```
  % setacl –D –u user::rw-,group::r--,other::r--,\
    user:jdoe:rw– foo
  ```

```
% setacl –d –u user::rw-,group::r--,other::r--,\
  user:wilson:rwx foo
```

Any directory that is created within the directory `foo` now inherits the following ACL as the access ACL as well as the default directory ACL:

```
#
# file: foo
# owner: smith
# group: system
#
user::rw-
user:jdoe:rw-
group::r--
other::r--
```

The following ACL would be inherited as the default access ACL:

```
#
# file: foo
# owner: smith
# group: system
#
user::rw-
user:wilson:rwx
group::r--
other::r--
```

Any file created in directory `foo` now inherits the following ACL as the access ACL:

```
#
# file: foo
# owner: smith
# group: system
#
user::rw-
user:wilson:rwx
group::r--
other::r--
```

At a minimum, each ACL contains three entries:

- One for the owning-user
- One for the owning-group
- One for the `other` entry

These entries correspond to the traditional permission bits for the file or directory. If ACLs are enabled and you use the `chmod` command to change the traditional permission bits of a file or a directory, `chmod` also makes the appropriate changes to the access ACL for the owning user, the owning group, and the other entry.

To change the group, use the `chgrp` command. If you do not own the file or if you do not belong to the new group, you must become superuser to change the group name or group ID. To change the owner, use the `chown` command. To change the ownership of a file, you must be superuser.

When a file or directory is created, the owner and group are set in the same manner as without ACLs. The owner is set to the owner of the process creating the file. The group is set to the group of the parent directory if the `mount` option `grpid` is set on the file system. If the directory is set `setgid`, then the directory's `gid` is always used. If the directory is not `setgid` and the `nogrpid` option is set, then the `egid` of the process is used.

5.8 Interaction of ACLs with Commands, Utilities, and Applications

ACLs are a POSIX and System V compatible extension to UNIX based on POSIX P1003.6 Draft 13. Not all existing commands, utilities, and applications properly use or propagate ACLs, especially applications that are not from Compaq or applications that are not POSIX compliant. If you use any command, utility, or application to access or manipulate a filesystem object (file or directory) that has an ACL, you must check the ACL after completion to make sure that the ACL has not been removed or changed.

Many programs that modify files use the following process:

- Create the new version of the file with a temporary name
- Delete the existing version of the file
- Rename the new version from the temporary name to the real name.

When the file being modified has an ACL and the program does not replicate the ACL when creating the temporary version of the file, the above procedure will delete the file's ACL, or replace it with the default access ACL of the parent directory (if it has one). If you use such a program on a file with an ACL, you have to restore the ACL afterwards. This procedure also causes any hard links to be removed from the file. Some common commands that use this method of modifying files are:

- `gzip`
- `compress`
- `emacs`

A workaround is to copy the original file to a temporary file, do any processing on the temorary file, an then use `cp` without the `-p` option to copy back. This procedure retains the original ACL.

Any time that you copy a file with an ACL, you should use the `cp -p` command. This will properly copy the ACL and any other extended attribute (property list).

For more information on writing or modifying programs to work properly with ACLs, see the `acl(4)` reference page and Chapter 21. For information on how ACLs interact with the archiving tools (`tar`, `pax`, and `cpio`), see Section 11.6. Until UNIX variants conform to a standard representation for ACLs, and the base utilities are converted to preserve ACLs, it is the user's responsibility to keep files protected. The permission bits on all newly created objects can be set by using umask or default ACLs. As with traditional UNIX discretionary file attributes, the burden of protecting files is on the user.

Note

Compaq recommends that you use restrictive traditional permissions, such as `other::---` and `group::---`, and then grant access to individual users with user entries. If an ACL is lost, unintended access is not allowed.

Part 2

Administrator's Guide to Security

6

Introduction for Administrators

The Tru64 UNIX operating system with the optional enhanced security subsets installed can be configured to meet the C2 security requirements as defined in the *Orange Book*. When the enhanced security subsets are installed and configured to meet C2 security requirements, the system is referred to as a trusted system. This chapter defines a trusted system and the requirements that the system was designed to satisfy. It introduces the terms and security concepts that are fundamental to system security, and it summarizes the major features of the system security policy enforced by the trusted system. This chapter also summarizes the major characteristics of the system, including its primary databases, subsystems, resource configuration files, and outlines the administrative roles and functions necessary to maintain a trusted system.

6.1 Frequently Asked Questions About Trusted Systems

When considering the use of a trusted system, some important questions are frequently asked:

- What is the performance impact of running a trusted system? Users are often concerned that enhancing a system's security features will hinder its usability by slowing down processing.
- Will a trusted system unnecessarily restrict an ordinary user's ability to accomplish their work?
- Can any UNIX system, including Tru64 UNIX, be secure? Users are sometimes skeptical that a system that has a reputation for being easy to penetrate can be used as the basis for a trusted system.

Although the trusted system has extended the Tru64 UNIX operating system to enforce additional security checks, the basic mechanisms of the system remain the same. Compatibility at the binary program interface and at the user interface have been design criteria for the trusted system. The trust enhancements have been made to incur as small a reduction in performance and as little unexpected system behavior as possible.

Although users will see few differences, the additional security requirements do add overhead for the administrative staff. Not only must this staff be familiar with the tasks involved in administering a trusted system, they

must also be familiar with the trusted system mechanisms so they can understand the implications of their actions.

A knowledgeable administrative staff contributes to the security of any site. In fact, training both the administrative staff and the users is one of the best ways you can protect the system against penetration.

6.2 Defining a Trusted System

A trusted system is one that employs sufficient hardware and software integrity measures to allow its use for simultaneously processing a range of sensitive or confidential information. A trusted system can be trusted to perform correctly in two important ways:

- The system's operational features – in particular, its application interface – operate correctly and satisfy the computing needs of the system's users.
- The system's security features enforce the site's security policy and offer adequate protection from threats.

A security policy is a statement of the rules and practices that regulate how an organization manages, protects, and distributes sensitive information. The system's security mechanisms maintain full compatibility with existing Tru64 UNIX security mechanisms while expanding the protection of user and system information.

An organization carries out its security policy by running the system as described in this manual and by adhering to the administrative and procedural guidelines defined for the system.

Understanding the concept of a trusted computing base (TCB) is important to understanding a trusted system. The TCB is the set of protection mechanisms that enforces the system's security policy. It includes all of the code that runs with hardware privilege (that is, the kernel) and all code running in processes that cooperate with the operating system to enforce the security policy. The system's TCB consists of the following parts:

- A modified Tru64 UNIX kernel. The kernel runs in the privileged execution mode of the system's CPU. The trusted system's kernel is isolated from the rest of the system because it runs in a separate execution domain – the processor's protected supervisor state.
- Trusted commands and utilities. The system corrects, modifies, and adds to the Tru64 UNIX software.

A TCB is typically defined in terms of subjects and objects. The TCB oversees and monitors interactions between subjects (active entities such as processes) and objects (passive entities such as files, devices, and inter-process communication mechanisms). See Appendix A for the software part of the system's TCB.

The trusted system protects a Tru64 UNIX system and its users against a variety of threats and system compromises. The most important of these threats are summarized in Table 6–1.

Table 6–1: Potential System Threats

Threat	Effect
Data disclosure	The threat of disclosure occurs when a user gains access to information for which that user does not have a need-to-know. Need-to-know restrictions are enforced by the system's discretionary access control features, which enable users, at their own discretion, to allow their information to be accessed by other users.
Loss of data integrity	The threat of integrity loss occurs when user or system information is overwritten – either intentionally or inadvertently. Loss of data integrity can occur from hardware failures (for example, disk failures) or software failures. When a loss of data integrity occurs, an opportunity is created for an unauthorized user to change information that affects the ability of the system to function properly.
Loss of TCB integrity	The TCB enforces the system's security policy. Any loss of integrity of TCB programs and files, including the executable copies of those programs in memory, constitutes a compromise of the integrity of the TCB itself and can lead to incorrect enforcement of the security policy.
Denial of service	To function usefully, the system must respond to requests for service. One way to compromise the usefulness of a system is to cause it to fail in its ability to process work. When denial of service occurs, users lose the ability to access their information. Depending upon the method of attack, the threat of denial of service can accompany any of the other previously mentioned threats.

6.3 Enhanced Security Features

The Tru64 UNIX operating system, with the optional enhanced security subset installed and in use, is designed to meet or exceed the requirements of the C2 evaluation class of Department of Defense 5200.28-STD as described in the *Orange Book*. The audit and ACL features can be enabled without the optional enhanced security subsets installed. The enhanced password features require the enhanced security subset to be installed.

6.3.1 Audit Features

Tru64 UNIX provides the following audit features:

- The ability to send audit logs to a remote host
- The following types of event auditing:
  - Site-defined support
  - System call support
  - Habitat support
  - Application support
- Fine-grained preselection of system events, application events, and site-definable events
- Extensive postreduction of system events, application events, and site-definable events
- Link-time configurability of the audit subsystem
- A per user audit characteristics profile (with enhanced I and A)
- OSF/Motif based interfaces

The audit system is set up using the audit configuration utility and maintained from the command line or with the `dxaudit` GUI.

6.3.2 Identification and Authentication (I and A) Features

Enhanced security provides the following I and A features:

- Password control
  - Configurable password length, up to 80 characters maximum.
  - Configurable password lifetimes. This includes an optional minimum interval between password changes.
  - A dynamic minimum password length, based directly on the Department of Defense *Password Management Guideline (Green Book)* guidelines and the password lifetime or a minimum length set by the system administrator.
  - Per-user password generation flags, which include the ability to require a user to have a generated password.
  - Recording of who (besides the user) last changed the user's password.
  - Configurable password usage history (0-9 previously used passwords).
- Login control
  - Optional recording of last terminal and time of the last successful login, and of the last unsuccessful login attempt.

  - Automatic account lockout after a specified number of consecutive bad access attempts. In cases of system database corruption, root can still log into the console (`/dev/console`).
  - A per-terminal setting for delay between consecutive login attempts, and the maximum amount of time each attempt is allowed before being declared a failed attempt.
  - A per-terminal setting for maximum consecutive failed login attempts before locking any new accesses from that terminal.
- Ownership for pseudoaccounts. This allows a way to differentiate auditable users when two `/etc/passwd` entries share a UID, such as `uucp` and `uucpa`.
- A notion of whether the account is "retired" or "locked." These are fundamentally the same as far as granting access is concerned, but are different administratively. There is also a provision for the auto-retirement of accounts by recording an expiration on the account itself.
- System default values for the various I and A fields. Most default values can be overridden on a per user basis. However, some values such as the password expiration warning time are system-wide and cannot be changed on a per user basis.

6.3.3 Access Control Lists (ACLs)

Traditionally, UNIX systems control a user's access to files and directories (file system objects) using a method of discretionary access control (DAC) normally referred to as the permission bits. By default, Tru64 UNIX systems are run using this untrusted method of DAC for file system objects.

ACLs provide greater granularity of file system object protection than the default DAC protection. The level of file system object protection provided by ACLs is required by trusted systems, but ACLs can be enabled separately from the other security options. This allows you to tailor your system to use only the security options that you need, instead of having to setup a fully trusted system.

6.3.4 Integrity Features

The enhanced security option provides the capability to validate the correct operation of hardware, firmware, and software components of the TCB. The firmware includes power-on diagnostics and more extensive diagnostics that can optionally be enabled. The firmware itself resides in EEPROM memory and can be physically write-protected. It can be compared with, or reloaded from, an off-line master copy. Compaq's service engineers can run additional hardware diagnostics as well.

The firmware can require authorization to load any operating software other than the default, or to execute privileged console monitor commands that examine or modify memory.

Once the operating system has been loaded, you can run system diagnostics that validate the correct operation of the hardware and software. In addition, test suites are available to ensure the correct operation of the operating system software.

You can use the following tools to detect inconsistencies in the TCB software and databases:

`fverify`	The `fverify` program reads subset inventory records from standard input and verifies that the attributes for the files on the system match the attributes listed in the corresponding records. Missing files and inconsistencies in file size, checksum, user ID, group ID, permissions, and file type are reported.
`authck`	The `authck` program checks both the overall structure and internal field consistency of all components of the authentication database. It reports all problems it finds.

6.3.5 Security db Utilities

A customized version of the Berkeley Database (Berkeley DB) is embedded in Tru64 UNIX to provide high-performance database support for critical security files. The DB includes full transactional support and database recovery, using write-ahead logging and checkpointing to record changes. In the event of catastrophic failure, the security database can be restored to its last transaction-consistent state by restoring the database files and rolling the log forward.

The following database management utilities are included with Enhanced Security:

`db_archive`	Displays the enhanced security database log files no longer involved in active transactions that can safely be backed up and deleted to regain space on `/var`.
`db_checkpoint`	Flushes memory, writes a checkpoint record to the log and flushes the log to disk.
`db_load`	Reads from a file or standard input and loads into a database.

`db_unload`	Unloads the database into a file.
`db_stat`	Displays the security database statistics.
`db_recover`	Restores the database to a consistent state after an unexpected failure.

In general, the security database is loaded or unloaded only by installation utilities. While the database has been designed to minimize database administration tasks, the addition of security database log files does present the possibility of log files expanding to fill `/var`. Thus, the security configuration utility includes an option that creates a `cron` job to periodically delete log files no longer involved in active transactions.

6.3.6 Common Data Security Architecture (CDSA)

The Common Data Security Architecture (CDSA) is a multiplatform, industry-standard security infrastructure that is available for Tru64 UNIX Version 5.1 or higher releases as an advanced developers kit (ADK). It provides a standards-based, stable programming interface that applications can use to access operating system security services, allowing developers to create cross-platform, security-enabled applications. Applications request security services, such as cryptography and other public key operations, through a dynamically extensible application programming interface (API). These requests are serviced by a set of plug-in security service modules (SPIs), which can be supplemented or changed as business needs and technologies evolve.

CDSA was originally developed by Intel Architecture Labs and was released to the Open Source community in May 2000. The Compaq CDSA implementation is based on the Intel Version 2.0 Release 3.11 reference platform, which implements CDSA Version 2 with Corrigenda, as defined in The Open Group's Technical Standard C914, May 2000.

The CDSA Advanced Developer's Kit (ADK) for Tru64 UNIX Version 5.1 or higher is available only from the following Compaq Web site at:

http://tru64unix.compaq.com/internet/download.htm

The kit contains all the components necessary to set up a cryptographic service. This software is free to all users. Follow the instructions on the Web site to download the kit.

6.3.7 Single Sign On (SSO)

Single Sign On (SSO) is an optional user authentication security feature available for the Tru64 UNIX operating system. Based on Kerberos technology, the Tru64 UNIX SSO software increases an organization's level of security and decreases user account maintenance and administration in a heterogeneous intranet.

The SSO software and documentation is found on the Associated Products CD-ROM included in your software distribution.

6.3.8 Additional Security Software

Several third party security products are made available for the Tru64 UNIX operating system on the Internet Express CD-ROM that comes with your AlphaServer hardware. A secure socket layer (SSL), the FireScreen firewall product, the RID Denial of Service Scanner, `tcpwrapper`, and other security-related software is included.

You can learn more about the these products and also order the CD-ROM on the Compaq Web site at http://tru64unix.compaq.com/internet/osis.htm.

More security software such as `tripwire`, `wuftpd`, `lsof`, `crack` is available either on the Freeware CD-ROM included in your software distribution or from public domain sites on the Internet.

6.4 Graphical Administration Utilities

The following graphical utilities help you deal with the day-to-day security administration on your local machine:

`dxaccounts`	The Account Manager in the Common Desktop Environment (CDE) or the `dxaccounts` program from the command line allows you to create and modify all user accounts, and to modify the system defaults. You can find the Account Manager under the Application Manager → System_Admin → System_Management_Utilities → Daily_Admin → Account Manager.
`dxaudit`	Use the `dxaudit` GUI to manage the audit system mask of events to audit. It can also be used to generate audit reports. The `dxaccounts` GUI can set a per user audit mask. Administrators have the flexibility to configure the audit subsystem without the requirement of installing additional security features.

	You can find the Audit Manager GUI under the CDE Application Manager → System_Admin → System_Management_Utilities → Daily_Admin → Audit Manager. The Audit Manager can also be started from the command line with the `/usr/tcb/bin/dxaudit` command.
`dxdevices`	Use the `dxdevices` program to configure devices. The Devices GUI is started from the command line with the `/usr/tcb/bin/dxdevices` command.
`dxsetacl`	Use the `dxsetacl` program to view and set ACLs on files and subdirectories. The `dxsetacl` GUI is started from the command line with the `/usr/tcb/bin/dxsetacl` command.

For more details about starting the GUIs from the command line, see the `dxaccounts`(8), `dxaudit`(8), and `dxdevices`(8) reference pages.

6.4.1 Installing and Configuring Enhanced Security

Before security can be configured, the enhanced security subsets (OSFC2SEC510 and OSFXC2SEC510) must be installed on your system. See the *Installation Guide* for more information.

System administrators can select the optional C2 security features that are required for their system. You do not have to configure all the features. The default security level consists of object reuse protection, traditional UNIX passwords, and discretionary access control; by running the `secconfig` command, you can select the security features appropriate for your system. The `secconfig` utility is found in CDE under Application Manager → System_Admin → System_Managent_Utilities → Configuration → Security. The `secconfig` utitlity can also be run from the command line.

The audit subsystem is configurable at kernel link time, regardless of the security level of the system. Auditing is initially configured using the `sysman auditconfig` utility. The identification and authorization (I and A) features are configured at boot time, so that the system administrator can configure the security level of the system. ACLs are enabled and disabled using the `secconfig` utility or the `sysconfig` command.

6.5 Administering the Trusted Operating System

An administrator of a trusted system is responsible for overseeing many additional security functions such as the following:

- Setting up security databases
- Monitoring the security and integrity of the system
- Auditing security-related events and maintaining the system's audit functions
- Performing miscellaneous administrative tasks associated with protected subsystems

6.5.1 Traditional Administrative Roles

An important difference between a nontrusted and a trusted Tru64 UNIX system is in the area of system administration. An effective administrator must understand the system's security policy, how it is controlled by the information entered into the system's security databases, and how any changes made in these databases affect user and administrator actions.

Note

On a trusted systemTru64 UNIX, the traditional security administrative roles may be performed by the same person. Trusted Tru64 UNIX does not support `sysadmin` and `isso` accounts. An administrator logged in as root can use the `dxaccounts`, `dxaudit`, and `dxdevices` interfaces to set up, modify, and maintain accounts and administer the security aspects of the system.

Administrators must be aware of the sensitivity of the information being protected at a site – the degree to which users are aware of, willing, and able to cooperate with the system's security policy, and the threat of penetration or misuse from insiders and outsiders. Only vigilance and proper use of the system can keep the system secure.

Table 6–2 summarizes these major roles and their associated responsibilities in the system. The sections that follow describe these responsibilities in greater detail.

Table 6–2: Traditional Administrative Roles

Role	Major Responsibilities
Information Systems Security Officer	Sets system defaults for users, maintains security-related authentication profile parameters, modifies user accounts, administers the audit subsystem, assigns devices, and ensures system integrity.

Table 6–2: Traditional Administrative Roles (cont.)

Role	Major Responsibilities
System administrator	Creates user accounts, creates and maintains file systems, and recovers from system failures.
Operator	Administers line printers, mounts and unmounts file systems, and starts up and shuts down the system.

Role association, coupled with sophisticated auditing features, enables a site to maintain accountability for administrative actions. This helps to prevent security problems and makes other problems easier to identify and solve.

6.5.1.1 Responsibilities of the Information Systems Security Officer

Note

On a trusted Tru64 UNIX system, responsibility for all of these traditional roles can be assumed by one person. An administrator with root privilege can perform any of the duties usually assigned to an ISSO.

The information systems security officer (ISSO) is primarily responsible for managing security-related mechanisms. The ISSO controls the way that users log in and identify themselves to the system. The ISSO must cooperate with the system administrator when performing security-related tasks; the system's checks and balances often require that each perform a separate part of a total task (for example, account creation). The following list describes specific ISSO responsibilities:

- Performs device assignment. Assigns devices (terminals, printers, and removable devices, such as floppy disk and magnetic tape). Specifies the appropriate operational parameters for these devices. (See Chapter 8.)
- Modifies user accounts. After accounts have been established by the system administrator, sets up authentication profiles reflecting the level of trust placed in those users. (See Chapter 9.)
- Audits system activity. Selects the security-relevant events that are to be audited by the system. Enables and disables auditing, sets audit parameters, produces reports, and regularly reviews audit data. (See Chapter 10.)
- Ensures system integrity. Ensures the integrity of the system by periodically running the `authck` program to check the integrity of the security databases and files critical to the correct operation of the system. (See the `authck`(8) reference page and Chapter 12.)

All of the ISSO functions, except integrity checking, can be performed using the Account Manager (`dxaccounts` interface). To perform ISSO functions, you must have root privileges and be logged on as root.

6.5.1.2 Responsibilities of the System Administrator

The system administrator is primarily responsible for account creation and disabling, and for ensuring the internal integrity of the system software and file systems. The system administrator also shares with the ISSO the responsibility for day-to-day user account maintenance.

The following list describes specific system administrator responsibilities:

- Creates user accounts. All accounts created by the system administrator have the default characteristics established by the ISSO for the system. Once an account has been created, the ISSO can modify that account, changing individual users' authentication profiles as appropriate. (See Chapter 9.)
- Creates groups. Creates new groups as part of user account creation. These groups are used by the system's discretionary access control mechanism. (See Chapter 9.)
- Modifies ISSO accounts. As an additional system security feature, the ISSOs are not authorized to modify their own authentication profiles. Instead, the system administrator performs this function. (See Chapter 9.)
- Creates file systems. Creates and maintains file systems by running programs such as `newfs` and `fsck`. See the *System Administration* manual and the `newfs`(8) and `fsck`(8) reference pages for details.
- Creates and maintains ACLs on the file systems. (See Chapter 11.)
- Restores the system files and users' files in the event of accidental deletion.

The system administrator creates user accounts and creates groups with the Account Manager (`dxaccounts`) interface.

To perform system administration functions, you must have root privileges and be logged on as root.

6.5.1.3 Responsibilities of the Operator

The operator is primarily responsible for ensuring that day-to-day hardware and software operations are performed in a trusted fashion.

The following list describes some specific operator responsibilities:

- Administers line printers. Enables and disables printers and performs other printer maintenance operations.

- Starts and shuts down the system. Boots the system, changes system run levels, and halts the system, when necessary.
- Mounts and unmounts file systems.
- Performs backups and file restorations.

To perform the operator functions, you must have root privileges and be logged on as root.

6.5.2 Protected Subsystems

Protected subsystems are collections of programs and resources that are grouped together by function and are important pieces of the TCB. They may or may not need privileges to accomplish their function. The system provides mechanisms for unified auditing within a protected subsystem. Administration of the includes performing subsystem administration tasks, and assuring proper installation and continued operation of the subsystem.

The components of a protected subsystem are protected with the group ID of the group allowed access rights to the programs and data in the subsystem. The only way for a user to access the subsystem information is by running programs in the subsystem.

The subsystem programs are set-group-ID (SGID) on execution to the subsystem's group. This method is also used in untrusted Tru64 UNIX systems. All of the subsystems have been modified to meet security and accountability requirements.

The system provides common mechanisms for implementing all of the protected subsystems, including the following:

- Ensuring that the subsystem databases are not corrupted
- Enforcing isolation between users
- Producing audit records

Table 6–3 summarizes the protected subsystems.

Table 6–3: Protected Subsystems

Database	Location	Contents
Protected password	`/tcb/files/auth.db` `/var/tcb/files/auth.db`	User authentication database
System defaults	`/etc/auth/system/default`	Default values for database fields
Terminal control	`/etc/auth/system/ttys.db`	Security information about each terminal

Table 6–3: Protected Subsystems (cont.)

Database	Location	Contents
File control	/etc/auth/system/files	Protection attributes of each system file
Device assignment	/etc/auth/system/devassign	Device-specific controls

6.5.2.1 Enhanced (Protected) Password Database

The protected password database stores the enhanced authentication profile for each user who has an account on the system. Each profile contains information such as the following:

- User name and ID
- Encrypted password
- User's audit characteristics
- Password generation parameters
- Successful and unsuccessful login times and terminals

The enhanced (protected) password database is located in the file /tcb/files/auth.db.

See the prpasswd(4) reference page for more information on the contents of the enhanced password database.

6.5.2.2 System Defaults Database

The system defaults database stores default values for database fields. These defaults are used when the administrator does not set explicit values in the enhanced (protected) password database, terminal control database, or device assignment database.

The system defaults database contains information such as the following:

- Default password generation parameters
- Default number of unsuccessful login attempts allowed per user
- Default number of unsuccessful login attempts allowed per directly connected terminal
- Default device assignment parameters

More information on the contents of the system defaults database located in /etc/auth/system/default can be found in the default(4) reference page.

6.5.2.3 Terminal Control Database

The terminal control database contains information that the administrator uses to control login activity at each terminal attached to the system. The system uses this database as an aid in controlling access to the system through terminals. The administrator can set different policies for logins at different terminals, depending upon the site's physical and administrative needs.

Each entry in the terminal control database contains information such as the following:

- Terminal device name
- User ID and time stamp of the last successful login attempt from this terminal
- User ID and time stamp of the last unsuccessful login attempt from this terminal
- Delay imposed between login attempts from this terminal
- Number of unsuccessful attempts that can be made before locking this terminal

When the system is installed, the terminal control database contains an entry for the system console. The ISSO modifies these initial values during system setup. A corresponding entry, also initially installed, is required in the device assignment database before logins are allowed.

For more information about the contents of the terminal control database located in `/etc/auth/system/ttys.db`, see the `ttys(4)` reference page. Procedures for adding terminals are described in Chapter 8.

6.5.2.4 File Control Database

The file control database contains information about the protection attributes of system files (that is, files important to the TCB's operation). This database helps maintain the integrity of the TCB. It contains one entry for each system file.

Each entry in the file control database contains the following information:

- Full pathname of the file
- File owner and group
- File mode and type
- Potential and granted privilege sets
- Access control list

When the system is installed, the file control database contains entries for all security relevant system files. The ISSO does not need to modify this database during system setup and rarely needs to update it during system operation. Chapter 12 describes how to check the integrity of the database and modify it if necessary.

For more information about the contents of the file control database located in `/etc/auth/system/files`, see the `files`(4) reference page.

6.5.2.5 Device Assignment Database

The device assignment database contains information about devices that are used to exchange data with users. Each login terminal must have an entry in the device assignment database. The system uses this database as an aid in restricting the security attributes of data that can be sent or received through the system's devices.

Each entry in the device assignment database contains information that describes a device and that relates the device pathname to the appropriate physical device. This is necessary because a number of distinct pathnames can refer to the same physical device. For example, two pathnames can refer to the same serial port – one with modem control enabled and the other with modem control disabled.

Each entry in the device assignment database contains information such as the following:

- Device pathname
- Other pathnames referencing the same physical device
- Device type

Entries referring to login terminals must have corresponding entries in the terminal control database.

The device assignment database is located in `/etc/auth/system/devassign`. See the `devassign`(4) reference page and Chapter 8 for details

6.6 Enhanced Security in a Cluster Environment

All the features of Tru64 UNIX enhanced security are available in a cluster environment. In some cases the setup and configuration is different than a noncluster environment. The security configuration procedure is also different depending whether enhanced security is being enabled on an already running cluster or whether the cluster is being installed. In all cases, enhanced security runs across all machines in the cluster and is seen as a single enhanced security environment also known as a common

security domain. See Appendix F for more information on enhanced security in a cluster.

6.6.1 Installation Time Configuration

If you are installing the operating system on the first member of cluster, you need to completely set up your security environment before installing the TruCluster Server software. The security configuration, as well as other configuration data, will be propagated to other members as they come up. Chapter 7 explains how to set up enhanced security on a system not in a cluster.

If you are installing the operating system on a member of a cluster other than the first system, the enhanced security environment will be inherited from the existing systems in the cluster when the TruCluster software is installed.

6.6.2 Postinstallation Configuration

If you are enabling enhanced security on systems already running in a cluster environment, you need to setup enhanced security from a single machine and then reboot every machine in the cluster.

7

Setting Up the Trusted System

This chapter lists the security-related tasks that must be completed after installation and before the trusted system is ready for general use, and refers to other chapters and to reference pages that explain how to accomplish the tasks.

7.1 Installation Notes

Before the enhanced authentication mechanism and other enhanced security features can be set up, the Tru64 UNIX installation or update must be completed and the optional enhanced security subsets (OSFC2SEC510 and OSFXC2SEC510) must be installed. If you plan to enable the password triviality checks, you also need to ensure that the OSFDCMTEXTxxx subset is installed.

The installation procedures for the optional security subsets are found in the *Installation Guide*.

After the security subsets are installed, you will see a message like the following:

```
Configuring "C2-Security " (OSFC2SEC510)

Configuring "C2-Security GUI " (OSFXC2SEC510)
```

The message refers to the installation process, not the security configuration and setup. The `secconfig` utility is used to configure or set up the enhanced authentication mechanism and ACLs. The audit subsystem is a kernel option and is set up with `auditconfig`.

7.1.1 Full Installation

A full installation of Tru64 UNIX (either advanced or basic) brings up the system with only a root account. Run the `secconfig` script before adding accounts.

7.1.2 Update Installation

If you are updating your system from a previous version of Tru64 UNIX, all user accounts and databases are preserved, and running the `secconfig` program converts them to the enhanced security format.

7.2 Segment Sharing

Because of the page table sharing mechanism used for shared libraries, the normal file system permissions are not adequate to protect against unauthorized reading. For example, user `joe` has the following shared library:

```
-rw------- 2 joe staff 100000 Sep 18 1997 /usr/shlib/foo.so
```

When this shared library is used in a program, the text part of `foo.so` may be visible to other running processes even though they are not running as user `joe`. Only the text part of the library, not the data segment, is shared in this way.

To disable all segmentation and avoid any unauthorized sharing, answer "yes" when `secconfig` asks if you want to disable segment sharing. The `secconfig` script reports when segment sharing is already disabled.

Note

Disabling segment sharing can cause excessive memory use.

7.3 Installation Time Setup for Security

Enhanced security is included on CDE's Installation Checklist and can be configured at installation time. When you select Security, the `secconfig` utility is run to configure the enhanced authentication mechanism (enhanced security) and the ACL subsystem. The audit subsystem is configured as a kernel option. (Use the `auditconfig` utility to complete the setup of audit.)

If you are installing Tru64 UNIX from a console, you will find the audit subsystem listed as kernel configuration option. It can be selected and built into the kernel during the initial system configuration. (Use the `auditconfig` utility to complete the setup of audit.) Run the `secconfig` utility to configure the enhanced authentication mechanism and ACLs.

Use the following procedure to set up enhanced security on a new system.

1. Verify that the enhanced security subsets (`OSFC2SEC`*xxx* and `OSFXC2SEC`*xxx*) are installed. If you are going to configure password triviality checks, the `OSFDCMTEXT`*xxx* subset also needs to be installed. If the subsets are not installed, install them, using the *Installation Guide* if you need more information.
2. Log in as root.
3. Run the interactive `secconfig` command and select ENHANCED security when prompted for a security level.

4. Bring down your system to single user and reboot (your shutdown message should inform users of the impending password changes).

The auditconfig(8) reference page describes how to set up audit. The acl(4) reference page describes the ACL implementation and Section 11.3 describes the Tru64 UNIX ACL setup.

7.4 The secconfig Utility

The secconfig utility is an interactive program that allows you to toggle the security level on your system between BASE and ENHANCED. You can run the program while the system is in multiuser mode. However, depending on the security features chosen, when secconfig is complete, you may need to change the security features, you must reboot your system.

Before you can run secconfig, you must load the enhanced security subsets onto your system.

7.4.1 Setup Questions

Before configuring security, you need to be prepared to answer the following questions:

- Do you want to disable segment sharing?
- Do you want to enable ACLs?
- Do you want to run auditconfig?

7.4.2 Invoking secconfig

Verify the security subset installation and invoke secconfig as follows:

```
# /usr/sbin/setld -i | grep SEC
OSFC2SEC510   installed   C2-Security (System Administration)
OSFXC2SEC510  installed   C2-Security GUI (System Administration)

# sysman secconfig

# shutdown -r now
```

7.5 Configuring Security Features

You can configure security features individually or you can enable all the security features.

7.5.1 Configuring Audit

You can run the audit subsystem without installing the security subsets. Configure the Audit Subsystem kernel option and then run the auditconfig utility to configure audit. The auditconfig utility includes

the kernel build procedures. See the auditconfig(8) and doconfig(8) reference pages for more information.

7.5.2 Configuring ACLs

You can run the ACL subsystem without installing the optional enhanced security subsets. ACL processing is now dynamically enabled and disabled using the sysconfig command or the secconfig utility. See the sysconfig(8) reference page and Section 11.3 for more information.

7.5.3 Configuring Enhanced Authentication with NIS

Running the secconfig command creates an enhanced user profile for each user on the system. If the user accounts are local, the passwords are expired and the users must enter a new password the next time they log in.

If the machine has a password database served by NIS (Network Information Service), secconfig asks if you want to create a local enhanced authentication profile for each user in the NIS server password database. If you do, see Chapter 9 for a description of how to distribute the enhanced authentication database with NIS. Subsequent changes in NIS passwords are not propagated to the database. The enhanced passwords now on the local machine are expired and users must enter a new password the next time they log in.

If you change the security level back to BASE security, the enhanced authentication profile files are left in place. When you return to ENHANCED security, as long as there is an enhanced authentication profile file and it contains a password, the enhanced password is updated.

You can use the edauth utility to view specified databases.

7.5.4 Authentication Features Configuration

Enhanced security provides the ability to specify system default values that apply to users, terminals, and devices. Thus, an administrator is not required to replicate values when they are all the same. The following sections briefly describe some common defaults and how you can configure them. The system defaults are stored in the default database at /etc/auth/system/default.

This database can contain four types of fields:

- System wide fields that exist only in the default database. These fields are prefixed with a d_.
- User default fields, whose values can be overridden by the corresponding fields in a user's profile. These fields are prefixed with a u_.

- Terminal control fields, whose values can be overridden by the corresponding fields in the terminal control database. These fields are prefixed with a t_.
- Device assignment fields, whose values can be overridden by the corresponding fields in the device assignment database file. These fields are prefixed with v_.

The dxaccounts GUI can modify the default fields for users by going to Local Templates→Default. The dxdevices GUI can modify the default fields for devices. The edauth utility provides a lower-level interface to all of the default fields.

See the authcap(4) reference page for a description of the file format and field values, the edauth(8) reference page for use of edauth, and the default(4), prpasswd(4), ttys(4), and devassign(4) reference pages for complete descriptions of the various fields and an interpretation of values.

Note

If you are going to configure password triviality checks, the OSFDCMTEXT*xxx* document extension subset (with the spell check libraries) needs to be installed.

7.5.4.1 Aging

If you do not want password aging on your system, in the default database set u_exp and u_life to 0, and then (because of the way the default methods of determining length restrictions on passwords work based on the password lifetime) also set u_minchosen and u_maxchosen to appropriate values for the site.

An example entry could be as follows:

```
:u_exp#0:u_life#0:u_minchosen#5:u_maxchosen#32:\
```

7.5.4.2 Minimum Change Time

You can remove the minimum change time interval by setting the u_minchg field to 0 as follows:

```
:u_minchg#0:\
```

This allows users to immediately change their password after a previous password change.

7.5.4.3 Changing Controls

The password-changing controls can be configured to your site's needs. By putting the following fields in the `default` database, you allow users to select how their passwords are chosen:

```
:u_pickpw:u_genpwd:u_genchars:u_genletters:u_restrict:\
:u_policy:u_nullpw:u_pwdepth#0:\
```

(Of those, `u_pwdepth` is numeric and the rest are Boolean. A Boolean field is true if it is specified and false if it is followed by an @.)

7.5.4.4 Maximum Login Attempts

In breakin detection, consective login failures are counted and compared to a maximum for a user (`u_maxtries`) or for a terminal (`t_maxtries`). If the maximum is exceeded, then logins to the user account or the terminal are disabled for a time period specified by `u_unlock` or `t_unlock`. To disable breakin evasion for user accounts, set `u_maxtries` to 0. To disable for terminals, set `t_maxtries` to 0. The `default` database entry for users would be as follows:

```
:u_maxtries#0:\
```

7.5.4.5 Time Between Login Attempts

If the default evasion time (86400 seconds or 24 hours) is not appropriate for your site, change the `u_unlock` field to an appropriate value for your site (number of seconds before a success is recognized after the last failure, once the `u_maxtries` limit is reached). Setting the `u_unlock` field to 0 (`:u_unlock#0:`) sets the time between login attempts to infinity (no automatic reenabling occurs). The equivalent behavior for terminals is controlled by `t_maxtries`.

7.5.4.6 Time Between Logins

You can set system wide maximum allowable time between logins in the `u_max_login_intvl` field of the `default` database.

The system default login timeout for terminals can be changed in the `t_login_timeout` field of the `default` database. It can also be set in the * entry of the `ttys` database. This field should be 0 (infinite) for X displays.

7.5.4.7 Per-Terminal Login Records

If you do not want to record per-terminal login successes and failures, set the `d_skip_ttys_updates` Boolean field in the `default` database as follows:

```
:d_skip_ttys_updates:\
```

This has the side effect of disabling any further per-terminal breakin evasion.

7.5.4.8 Successful Login Logging

Strict C2 security requires the logging of successful logins. To disable this logging, set the `d_skip_success_login_log` Boolean field as follows:

```
:d_skip_success_login_log:\
```

7.5.4.9 Failed Login Logging

Failed login attempts to user accounts are normally recorded. To disable this logging, which also disables breakin detection and evasion system wide, set the `d_skip_fail_login_log` Boolean field as follows:

```
:d_skip_fail_login_log:\
```

7.5.4.10 Automatic Enhanced Profile Creation

Setting the `d_auto_migrate_users` Boolean field allows the creation of enhanced profiles at login time if they are missing, so that traditional methods of adding user profiles can be used without change.

7.5.4.11 Vouching

You can set the `d_accept_alternate_vouching` field to allow enhanced security and DCE to work together.

7.5.4.12 Encryption

If you want the user passwords to stay in the `/etc/passwd` file to support programs that use `crypt()` to do password validation, but still want to use other features of enhanced profiles, put the following entry in the `default` database before running `secconfig`:

```
:u_newcrypt#3:\
```

This corresponds to the AUTH_CRYPT_C1CRYPT value from the `<prot.h>` file.

7.6 System Administrator Tasks

On a Tru64 UNIX system the `root` account is used to perform both system administration and ISSO tasks. The system administrator traditionally performs the following tasks using the Account Manager (`dxaccounts`) program:

- Creates groups.
- Creates accounts for users.

- Verifies that the file systems containing users' home directory are mounted. You do not need to create the directories themselves.

See Chapter 9 and the `dxaccounts`(8) reference page for more information.

7.7 ISSO Tasks

On a Tru64 UNIX system the `root` account is used to perform both system administration and ISSO tasks. The ISSO traditionally performs the tasks described in the following sections using the Account Manager (`dxaccount` program).

7.7.1 Check System Defaults

The ISSO checks that the following general defaults and account defaults conform to the site's security policy:

- The user password policy (whether users can pick their own passwords, what type of passwords the system generates, and so on)
- The login controls for accounts, such as the maximum number of unsuccessful attempts

7.7.2 Modifying a User Account

The ISSO modifies the accounts of any users who have fewer restrictions or more restrictions than the defaults.

If users' accounts are locked by default when they are created, you need to unlock the accounts before users can login. Depending on the procedures established at your site, you may want to unlock all accounts when created or unlock them when the users are ready to login for the first time.

See Chapter 9 for more information.

7.7.3 Assigning Terminal Devices

Use the `dxdevices` program to perform the following device-assignment tasks:

- Sets the device defaults.

If terminal devices are locked by default, you need to unlock them before users can login.

See Chapter 8 and the `dxdevices`(8) reference page for more information.

7.7.4 Setting Up Auditing

The ISSO performs the following tasks to set up the audit system:

- Specifies whether auditing is enabled or disabled when the system boots.
- Specifies which events should be audited.

See Chapter 10 for more information.

7.8 Backing Up the System

Make a backup copy of the root file system as a precaution. All the files that have been modified during system setup will be copied.

The backup can be made by using one of the following commands (`dump` only works on UFS file systems):

```
# dump -0uf /dev/rmt0h /
```

or

```
# vdump -0Nuf /dev/rmt0h /
```

Substitute the appropriate tape device for your system.

8

Creating and Modifying Secure Devices

The Information System Security Officer (ISSO) is traditionally responsible for assigning the devices that are included in the system's trusted computing base (TCB) and for defining the security characteristics of those devices. On a Tru64 UNIX system root access is required to assign devices. The trusted Tru64 UNIX system supports terminals as part of the TCB. This chapter describes how to define those devices in a secure system.

8.1 Defining Security Characteristics

The ISSO traditionally defines the security characteristics of all the terminals that are part of the system using the `dxdevices` program. To do this, the ISSO performs the following tasks:

- Creates and maintains device-specific information. The ISSO can override system defaults for an individual device, where appropriate, to grant additional rights or to impose additional restrictions. The ISSO can also lock a terminal to prevent use.
- Sets default control parameters for the devices that are included in the system's secure configuration. The system defaults for terminals are as follows:
  - Maximum number of unsuccessful login attempts is 10.
  - Login timeout as shipped is unset, which implicitly defaults to 0 which is treated as infinite.
  - Delay between unsuccessful login attempts is 2 seconds.

The ISSO is usually responsible for ensuring that all device assignments, whether they are set explicitly or by default, conform to a site's security requirements.

Before you create or modify a secure device, all of the typical device installation procedures required during ordinary system hardware and software installation must be completed. The special files for devices must exist in the `/dev` directory and have the appropriate permissions. The special files for terminals must be owned by `root`, have the group set to `tty`, and have the mode set to `0620`.

You can verify that the installation has been completed with the `ls` command. The following example is typical:

```
# ls –lg /dev/tty*

crw----------  1  root  tty  0, 2 Aug 15 09:29 /dev/tty00

crw----------  1  root  tty  0, 3 Aug 15 09:29 /dev/tty01
```

8.1.1 Modifying, Adding, and Removing Devices with the dxdevices Program

Using the Devices dialog box, select the Modify/Create dialog box then the Select devices dialog box. To add or remove a device, first select or enter the device, then click on File to make the required changes. To modify a device, first select the device, then click on Modify to make the required changes. See the online help for `dxdevices` for more information.

8.1.2 Setting Default Values with the dxdevices Program

Using the Devices dialog box, select the Defaults dialog box. Set the system defaults for all of your terminals as required. A terminal uses these defaults unless specifically overridden by settings in the Modify Terminal dialog box. See the online help for `dxdevices` for more information.

8.2 Updating Security Databases

When you assign device defaults or device-specific parameters, the system updates the following security databases:

- The system defaults database, `/etc/auth/system/default`, contains the default values (for example, default control parameters) for all system devices.
- The device assignment database, `/etc/auth/system/devassign`, contains device-specific values for system devices.
- The terminal control database, `/etc/auth/system/ttys.db`, contains device-specific values for authentication (for example, the number of failed login attempts).

Each device to be used in your secure configuration must have an entry in the device assignment database. This database centralizes information about the security characteristics of all system devices. It includes the device pathname and type. By default a wildcard entry exists for terminals (but not X displays) in the `/etc/auth/system/ttys.db` and `/etc/auth/system/devassign` databases.

The X display entries shipped on the system have `:t_login_timeout#0:` entries in them, in case a site changes its system default login timeout. If wildcard X display entries are needed, they can be created as follows:

```
# echo \
 \'*\:*:t_devname=*\:*:t_login_timeout#0:t_xdisplay:chkent:\' \
                                          | /tcb/bin/edauth -s -dt

# echo \'*\:*:v_type=xdisplay:chkent:\' | /tcb/bin/edauth -s -dv
```

9

Creating and Maintaining Accounts

Accounts are created and maintained on a system using the Account Manager (`dxaccounts`) GUI or the `useradd`, `usrmod`, `userdel` command-line utilities. This chapter describes how to create and maintain local accounts under enhanced security. It also describes how to configure the Network Information Service (NIS) and how to use NIS to create and maintain distributed accounts under enhanced security.

9.1 Authentication Subsystem

The authentication subsystem verifies that users who log in to the system have the required password. It is the framework in which processes, protected subsystems, and the kernel work together to ensure that only authorized users and their processes gain access to the system.

The system administrator is responsible for ensuring that all user authorizations, whether they are set explicitly or by default, conform to a site's security requirements.

The authentication subsystem uses and maintains the following security databases. These databases contain parameters and statistics for the system, for users, and for terminals. For a summary of the contents of these databases, see Chapter 17 and the appropriate reference pages:

- Password file (`passwd(4)`)
- Protected Password database (`prpasswd(4)`)
- System Defaults database (`default(4)`)
- Terminal Control database (`ttys(4)`)
- File Control database (`files(4)`)
- Device Assignment database (`devassign(4)`)

9.1.1 Local User Account Databases

The following sections describe the local (not associated with NIS) authentication databases.

9.1.1.1 Local Database: Base Security

Base (BSD) security is the traditional level of security that is available on UNIX systems. Tru64 UNIX is configured in base security by default. The local base user account files are `/etc/passwd` and `/etc/group`. The data in these files is used to allow or deny a user access to the system and to files on the system.

Each line of `/etc/passwd` contains information about one user account. An entry contains the user name, UID, password, shell, and user identity information (traditionally referred to as GECOS data).

The `passwd` command changes a user's base password. The `useradd`, `userdel`, and `usermod` commands are used by the system manager to add and change user account information. The `vipw` command, which performs some consistency checks, can be used to directly edit the `/etc/passwd` file.

The `/etc/group` file contains group information. The `groupadd`, `groupdel`, `groupmod`, and `groups` commands are used to manipulate local base group information.

9.1.1.2 Local Database: Enhanced Security

Enhanced security is a Tru64 UNIX option that provides many additional security features for user accounts. It is configured using the `secconfig` utility after installing the optional enhanced security subsets. A system running enhanced security has a local user account database in addition to `/etc/passwd`. This database, sometimes called the enhanced (protected) password database, is composed of `/tcb/files/auth.db` and `/var/tcb/files/auth.db`. The `/tcb/files/auth.db` database contains accounts such as root that must be accessible in single-user mode, while `/var/tcb/files/auth.db` contains the majority of accounts.

The database has an entry for each user account defined in `/etc/passwd`. Under enhanced security, `/etc/passwd` remains unchanged except for the encrypted password, which moves from `/etc/passwd` into `auth.db`. The other fields in the `/etc/passwd` file (shell, GECOS information, and so forth) remain in `/etc/passwd` and are used in a normal fashion.

The enhanced security user account database uniquely identifies a user by username and UID, which must match the user's `/etc/passwd` entry. In addition to the encrypted password, an entry contains a set of fields and values used only by enhanced security. The `prpasswd(4)` reference page describes these fields, and the `authcap(4)` reference page describes the file format.

A user account can be associated with a template account, which can be used to specify default values for a group of users. An account is always finally

associated with the system default template values that are contained in the /etc/auth/system/default file.

The passwd command changes a user's password under enhanced security. The dxaccounts program or the useradd, usermod, and userdel commands are used by the system manager to add, change, and delete user account information.

9.1.1.3 Templates for User Accounts

A user's entry in the enhanced security user account database is called his profile. Security-aware programs interpret the fields and values in a profile. A user profile need not contain every possible field. If a field is not specified in a user's profile, the system looks in the template account associated with the user, and finally in the system default template, until it finds a value for the field.

Values are obtained as follows:

- If the user profile contains a user-specific value, that value is used.
- If the user profile contains a reference to a template account, and no user-specific value is defined, the value in the template account is used.
- If neither the user profile nor the template account defines a value for a field and the system default template defines a value for that field, the system default template value is used.
- If the value is defined nowhere else, a static system default value is used for the field.

The system default template values are located in the /etc/auth/system/default file and can be modified using the dxaccounts View Local Template option, or with the edauth utility. Other template accounts are stored in auth.db. Note that template accounts have no corresponding /etc/passwd entry.

9.1.2 Distributing User Account Databases with NIS

The following sections review the account databases and their relationships under NIS.

9.1.2.1 Distributed Databases: NIS and Base Security

NIS can be used to distribute all or part of the base user account database to systems across the network. With NIS and base security you have two user account databases:

- The local base user account database in /etc/passwd and /etc/group.

- The NIS-distributed, base user account database is generated from the /var/yp/src/passwd and /var/yp/src/group files located on the NIS master server. These files, called NIS maps, are distributed in ndbm or btree format.

The entries in the NIS-distributed base user account database have the same fields as the /etc/passwd file entries.

A user's account information may be partially distributed. If the user's entry in the /etc/passwd file has a leading plus sign (+), both databases are read, but the information from the /etc/passwd file (except for the UID and GID fields) overlays the information from the NIS distributed user account database. The /etc/passwd file on each client system must contain a +: as the last entry in the file to allow users from the NIS distributed base user account database to log in. See Table 9–1 for a complete list of the /etc/passwd overrides.

Table 9–1: Controlling NIS with Local /etc/passwd Overrides

Symbol	Description
+:	If a user is not found in the local file, authenticate using the NIS file.
+username	Local file field overrides NIS. Used for partial distribution.
-username	User is excluded from all matches by local control.
+@netgr:	List of users to authenticate using the local file. See the netgroup(4) reference page.
-@netgr:	List of users to refuse using the NIS file. See the netgroup(4) reference page.
+:*:	Sends all password requests to the NIS map.

The passwd command changes the password in the local base user account database only. The NIS-distributed password is changed with the yppasswd command.

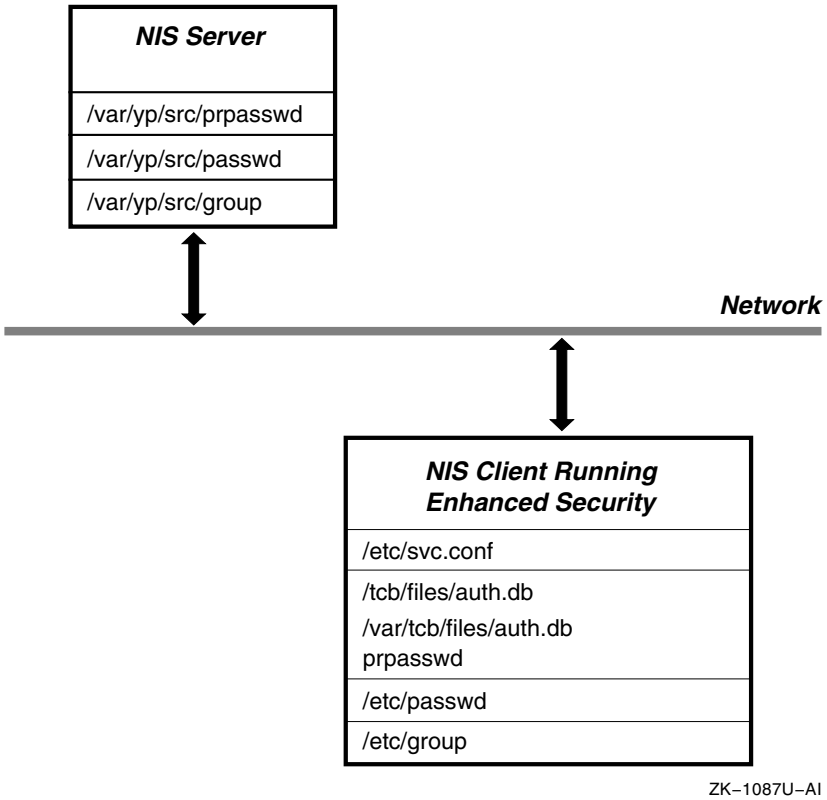
NIS user accounts can be modified using the dxaccounts View NIS User option, or by specifying the -x distributed=1 local=0 options to the useradd, usermod and userdel utilities. In addition, the system administrator can modify the NIS map source files in /var/yp/src and manually rebuild the maps using the makefile in /var/yp.

9.1.2.2 Distributed Databases: NIS and Enhanced Security

NIS can be used to distribute part or all of the enhanced security user account database, as well as part or all of the BSD user account database.

When you are running NIS over enhanced security you have four user account databases:

- The local base user account database in `/etc/password` and `/etc/group`.
- The NIS-distributed base user account database generated from the `/var/yp/src/passwd` and `/var/yp/src/group` files on the master server and distributed as `ndbm` or `btree` maps.
- The local enhanced security user account database.
- The NIS-distributed enhanced security user account database generated from the `/var/yp/src/prpasswd` file on the master server and distributed as `btree` maps.



The `auth=` entry in the `/etc/svc.conf` file indicates the order in which the local and NIS enhanced security user account databases are searched for user entries, either local first or NIS (yp) first.

The plus sign (+) override feature for `/etc/passwd` entries works as usual.

Note

When upgrading from a base security system with NIS to an enhanced security system, the `secconfig` utility only creates `auth.db` entries for NIS users (the `+username` entries in the `/etc/passwd` file) if you answer yes to the Create Entries for NIS Users question.

There is no override feature for the enhanced security user account database. A user's profile is contained completely in either the local database or in the NIS distributed data base. Although templates can be defined for NIS accounts and distributed as part of the NIS enhanced security maps, NIS does not distribute the system default template (`/etc/auth/system/default`). This template provides the final default values for fields not specified in a user's profile. Therefore, under enhanced security, a NIS client uses its own `/etc/auth/system/default` file to obtain final default values for both local and NIS user profiles. If the client system default file contains different values than that of the NIS master, unintended behavior can occur.

The `passwd` command changes the password in a user's local or NIS enhanced security entry. The `yppasswd` command changes the fields in the NIS-distributed base user account database as usual.

NIS user accounts can be modified using the `dxaccounts` View NIS User option, or by specifying the `-x distributed=1 local=0` options to the `useradd`, `usermod`, and `userdel` utilities.

9.1.2.3 Templates for NIS Accounts

The `/var/yp/src/prpasswd` file is the source for enhanced security user accounts distributed by NIS. It can contain template profiles as well as normal user profiles. As with a local user profile, a NIS user profile need not contain every possible field. If a field is not specified in a NIS user's profile, the system looks in the NIS template account associated with the user, and finally in the local system default template, until it finds a value for the field.

Values are obtained as follows:

- If the user profile contains a user-specific value, that value is used.
- If the user profile contains a reference to a template account, and no user-specific value is defined, the value in the template account is used.
- If neither the user profile nor the template account defines a value for a field and the system default template defines a value for that field, the system default template value is used.

- If the value is defined nowhere else, a static system default is used for the field.

NIS template accounts are modified using the `dxaccounts` View NIS Template option, or with the `edauth` utility.

The system default template values are located in the `/etc/auth/system/default` file on the NIS client. Note that NIS does not distribute the system default template. A NIS client uses its own `/etc/auth/system/default` file to obtain final default values for both local and NIS user profiles. If the client system default file contains different values than that of the NIS master, unpredicted behavior can occur.

9.2 Using dxaccounts for User Account Administration

The traditional role for the system administrator, as it relates to accounts, is to create and retire all user accounts, to create groups, and to modify the account templates. On a trusted Tru64 UNIX system, the `dxaccounts` program is used to create and maintain NIS and local user account databases.

9.2.1 Creating Local or NIS Groups

To create a group, use the `dxaccounts` program and proceed as follows:

1. Click on the View menu.
2. Select either Local Groups or NIS Groups from the popup menu.
3. Click on the Add icon on the toolbar and add the new group.

9.2.2 Creating Local or NIS User Accounts

Use the `dxaccounts` program to create user accounts. Click on the View menu item on the menu bar, select either Local Users or NIS Users from the popup menu, click on the Add icon on the toolbar. To create many accounts in a single session, fill in the information for a new user and provide a password, then click on Apply to create the account. Then fill in the information and provide a password for next user and once again click on Apply to create the account.

By default, new accounts are created in a locked state. If the account is not unlocked, the new user will receive an Account Disabled message when he or she tries to log in and the login attempt fails. To avoid this, the account can be explicitly unlocked when it is created. Optionally, it can be unlocked at a later time using `dxaccounts`.

9.2.3 Retiring Local or NIS Accounts (Enhanced Security Only)

To retire a user account, use the `dxaccounts` program. Click on the View menu, select either Local Users or NIS Users from the popup menu, select a user account from the Current View menu, and click on the Retire icon on the toolbar to retire the user account.

User names and UIDs associated with retired accounts cannot be reused. To delete, rather than retire, an account under enhanced security, manual intervention is necessary. Use `vipw` to remove the `/etc/passwd` entry for the account, then use the `edauth` utility to delete the `auth.db` entry.

9.2.4 Deleting Local or NIS Accounts (Base Security Only)

Run `dxaccounts` and click on the View menu item on the menu. Select either Local Users or NIS Users from the popup menu. Select a user account from the Current View menu and click the Delete icon on the toolbar. This permanently removes the account from the user account database.

9.2.5 Modifying the Local or NIS Account Template

An account template is used to establish default values for unspecified account parameters. There are three types of account templates: NIS templates, local templates, and the default template (which `dxaccounts` groups with the local templates). When an account is created, it is assigned the default template.

To modify an account template, use the `dxaccounts` program. Click on the View item from the menu bar and select either Local Templates or NIS Templates from the popup menu. Then double click on the desired template in the Current View.

9.2.6 Modifying Local or NIS User Accounts

To modify an account, use the `dxaccounts` program. Click on the View item from the menu bar and select either Local Users or NIS Users from the popup menu. Then double click on the desired user account in the Current View.

9.3 Using Commands for User Account Administration

The `useradd`, `userdel`, and `usermod` commands can perform most required user account administrative functions from the command line or a script. The `groupadd`, `groupmod`, and `groupdel` commands provide the same functions for groups. All of these commands share a set of defaults which the user may modify. Two of the defaults control whether the commands act on local or NIS user accounts. The `-D` option to `useradd`, `usermod`, or

groupadd is used to permanently change the default behavior of all the listed commands.

The -x option can be used with useradd, userdel, usermod, groupadd, groupmod, and groupdel commands to override the default local versus NIS behavior. The -x distributed=1 option causes the command to make changes on the NIS account, while the -x local=1 causes changes only on the local account.

The reference pages for these utilities describe the available options. The following sections primarily provide examples of common operations.

9.3.1 Creating Local or NIS Groups

To create a local group, enter the following:

```
# groupadd -g gid new_group_name
```

To create a NIS group, enter the following:

```
# groupadd -g gid -x distributed=1 new_group_name
```

9.3.2 Creating Local or NIS User Accounts

To create a local account with a specific UID (rather than a system-assigned UID), create a home directory, and after you are prompted for a password, enter the following:

```
# useradd -u uid -m -p new_user_name
New password:
Retype new password:
```

To create and unlock a local account, enter the following:

```
# useradd -x administrative_lock_applied=0 new_user_name
```

To create a NIS account with the next available UID, enter the following:

```
# useradd -x distributed=1 new_user_name
```

By default, new accounts are created in a locked state. If the account is not unlocked, the new user receives an Account Disabled message when they try to log in and the log in attempt fails. To avoid this, an account can be explicitly unlocked when it is created as shown in the previous example.

9.3.3 Retiring Local or NIS Accounts (Enhanced Security Only)

With enhanced security, accounts are retired rather than deleted. The user names and UIDs associated with retired accounts may not be reused. The same actions are performed whether or not the retire (-R) option is specified.

To retire a local account, enter the following:

```
# userdel -R user_name
```

To retire a NIS account, enter the following:

```
# userdel -R -x distributed=1 user_name
```

To truly delete, rather than retire, an account under enhanced security, use `vipw` to remove the `/etc/passwd` entry for the account, then use the `edauth` utility to delete the `auth.db` entry.

9.3.4 Deleting Local or NIS Accounts (Base Security Only)

The user names and UIDs associated with a deleted account are removed from the `/etc/passwd` file.

To delete a local account, enter the following:

```
# userdel user_name
```

To delete a NIS account, enter the following:

```
# userdel -x distributed=1 user_name
```

9.3.5 Modifying Local or NIS User Accounts

To change the shell of a local account, enter the following:

```
# usermod -s /bin/csh existing_user_name
```

To change the password of a local account, enter the following:

```
# usermod -p existing_user_name
New password:
Retype new password:
```

To change the shell of a NIS account, enter the following:

```
# usermod -x distributed=1 -s /bin/csh existing_NIS_user_name
```

9.4 Other Commands Associated with User Account Administration

You can use the `edauth` and `convuser` commands to view and modify various security databases. These commands are not intended for routine maintenance. See the appropriate reference pages for detailed information.

`edauth` — The `/usr/tcb/bin/edauth` utility can display and modify the various enhanced security databases, including the system default database and the user account database. Note that it does not affect the `/etc/passwd` file.

`convuser`	The `/usr/tcb/bin/convuser` utility is not intended for general use. It performs mass conversions of user profiles from base to enhanced form. While the `convuser` utility can also be used to attempt to revert the user profiles from enhanced form to base form, passwords are not necessarily compatible and may require changing. This command is typically used only by an update installation and by the `secconfig` utility.

9.5 NIS and Enhanced Security

You can use the Network Information Service (NIS) to centralize the management of the normal password group information and the enhanced user profiles maintained by enhanced security in the enhanced (protected) password database. A NIS master server can serve a mix of NIS clients, including ULTRIX and Tru64 UNIX systems (with and without enhanced profiles), and other manufacturer's systems with ordinary UNIX passwords and groups. NIS is documented in the *Network Administration: Services* manual.

The following sections describe the NIS configuration that specifically affects enhanced security.

9.5.1 Setting Up a NIS Master with Enhanced Security

If NIS is running on the master server, you must stop NIS using the `/sbin/init.d/nis stop` command, then take the following steps.

1. Ensure that Tru64 UNIX Version 5.1A or higher is installed.
2. Install the security subsets and set up security. See Chapter 7 for details.
3. Modify the system default template using the following command:

   ```
   # edauth -dd default
   ```

   Set the following fields:

   ```
   d_skip_success_login_log:
   d_skip_ttys_update:
   ```

4. Create `/var/yp/src/hosts`, `/var/yp/src/passwd`, `/var/yp/src/group`, and `/var/yp/src/prpasswd`. The files can be empty, but should exist before you run `sysman nis`.

5. Run the `sysman nis` program.

   a. When the `sysman nis` program first prompts for security (`-s` option to `ypbind`), choose `y` to run `ypbind -s`, which specifies a secure socket.

   b. When the `sysman nis` program again prompts for security (`-S` option to `ypbind`), choose `y` and specify a domain name and up to four authorized slave servers.

6. Make sure that the `/etc/svc.conf` file has the following entry:

   ```
   auth=local,yp.
   ```

7. Start NIS using the `/sbin/init.d/nis start` command.

9.5.1.1 Manual Procedure: Maps for Small User Account Databases

For a NIS master server supporting clients using enhanced security, a manual procedure is best. Set up the account maps using the `dxaccounts` program or alternatively the `adduser`, `addgroup`, `useradd`, `userdel`, and `usermod` commands. See Section 9.5.4 for another method of setting up accounts.

9.5.1.2 Automated Procedure: Maps for Large User Account Databases

If you have a large existing NIS distributed base user accounts database, you can automate the creation of the NIS distributed enhanced (protected) password database by entering the following command:

```
# convuser -Mc
```

Alternatively, you can create the map by creating a `/var/yp/src/prpasswd` file and then executing the following commands:

```
# /usr/tcb/bin/edauth -Lg > /var/yp/src/prpasswd
# cd /var/yp;  make prpasswd
```

9.5.2 Setting Up a NIS Slave Server with Enhanced Security

If NIS is running on the slave server, you must stop NIS using the `/sbin/init.d/nis stop` command. The following setup information is specific to a NIS slave server supporting clients using enhanced security:

1. Ensure that Tru64 UNIX Version 5.1 or higher is installed.
2. Install the security subsets and set up enhanced security. See Chapter 7 for details.
3. Modify the system default template using the following command:

   ```
   # edauth -dd default
   ```

Set the following fields:

```
d_skip_success_login_log:
d_skip_ttys_update:
```

4. Run the `sysman nis` program.

   a. When the `sysman nis` program first prompts for security (`–s` option to `ypbind`), choose `y` to run `ypbind –s`, which specifies a secure socket.

   b. When the `sysman nis` program again prompts for security (`–S` option to `ypbind`), choose `y` and specify a domain name and up to four authorized slave servers.

5. Edit the `/etc/svc.conf` file to include a `yp` entry for `auth`. The entry should be as follows: `auth=local,yp`.

6. Edit the `/var/yp/ypxfr_1perday`, `/var/yp/ypxfr_1perhour`, `/var/yp/ypxfr_2perday` files to add the following lines to each:

```
ypxfr -a "$method" prpasswd
ypxfr -a "$method" prpasswd_nonsecure
```

7. Start NIS using the `/sbin/init.d/nis start` command.

9.5.3 Setting Up a NIS Client with Enhanced Security

If NIS is running on the slave server, you must stop NIS using the `/sbin/init.d/nis stop` command. The following setup information is specific to a NIS client using enhanced password security:

1. Ensure that Tru64 UNIX Version 5.1 or higher is installed.
2. Install the security subsets and set up enhanced security. See Chapter 7 for details.
3. Modify the system default template using the following command:

```
# edauth -dd default
```

   Set the following fields:

```
d_skip_success_login_log:
d_skip_ttys_update:
```

4. Run the `sysman nis` program.

   a. When the `sysman nis` program first prompts for security (`–s` option to `ypbind`), choose `y` to run `ypbind –s`, which specifies a secure socket.

   b. When the `sysman nis` program again prompts for security (`–S` option to `ypbind`), choose `y` and specify a domain name and up to four authorized slave servers.

5. Edit the /etc/svc.conf file to include a yp entry for auth. The entry should be as follows: auth=local,yp.
6. Start NIS using the /sbin/init.d/nis start command.

9.5.4 Moving Local Accounts to NIS

To move existing local accounts to NIS, use the following command:

```
# edauth -Lg | edauth -NsC
```

9.5.5 Removing NIS Support

If you need to remove the NIS support from a trusted client system, copy the NIS accounts to the local database and then remove NIS using the following commands on the client:

```
# edauth -gN | edauth -sLC
# sysman nis
<select the Remove option from the menu>
```

The enhanced (protected) password database on the client machine is updated with any accounts from the NIS database that are not present in the local database.

9.5.6 Implementation Notes

The following information is specific to enhanced security and NIS:

- To change your password when running NIS with enhanced security, use the passwd command for both local and distributed enhanced (protected) password database entries. The passwd command uses the search list in the svc.conf file (auth=local,yp entry) and updates the password in the first enhanced (protected) password database entry it finds for the specified user, even if that entry is in the NIS-distributed enhanced password database.
- It is very important that each enhanced password database entry exists in only one database, either the local enhanced password database or the NIS-distributed enhanced password database. The routines that check and manipulate the enhanced password database information work on the first copy found (as defined in the svc.conf file). NIS yp routines work on the NIS-distributed enhanced password database only. This can cause confusing results if you have the same entry in both places. If this happens, delete one of the copies.
- It is strongly recommended that you do not distribute root account information. Maintaining a local root account on a client system allows

you to still log in on the client systems using the root account if your NIS server is down.

- Strict C2 security rules require an update to a user's enhanced profile each time that the user logs in, to maintain the last successful login information. On a NIS master, this requires rebuilding the map and shipping it to the slaves. Tru64 UNIX Version 5.1A makes these updates optional. The d_skip_success_login_log system default field controls this behavior, and Compaq recommends setting it to true.
- Although the user account database can only be modified on the NIS master server, disabling successful login logging means that the NIS master server does not always have to be available for logins to be successful if there is a properly configured NIS slave server.
- Scalability improvements include:
  - An update to a single entry does not always cause a rebuild of the entire prpasswd map. The map entries are updated directly if possible.
  - If successful login logging is enabled, a successful login does not wait for the NIS map to be distributed before completing. It only waits to make sure that the NIS master has been updated. If unsuccessful login logging is enabled, unsuccessful login attempts still wait for the map to be distributed to the slave servers before completing. This is required for security and timing issues.
- The database format for NIS maps is configurable. You can choose btree or hash in addition to ndbm. When using ndbm for NIS map storage, there is a limit to the number of account records that can be stored, which depends on the mix of account names and UIDs. A typical limit is about 30,000 entries, but some mixes of account names and UIDs can result in a limitation of fewer than 10,000 entries. Because of this constraint in ndbm, Compaq recommends that you use btree as your database format, especially when using enhanced security.
- NIS servers work best with a common database format. If a slave server has defined a different format than the master (ndbm instead of btree, for example), the time it takes to push any maps to that slave server is drastically increased because the slave server must rebuild its database one element at a time, instead of receiving the database from the master as a single entity.
- NIS slaves that are not listed in the ypservers NIS map on the NIS master can cause performance problems for NIS clients bound to those slaves. To solve this, define all NIS slaves in the ypservers NIS map on the NIS master. Then, on the slave server, execute the following commands to pull the user account databases from the NIS master:

```
# /var/yp/ypxfr -d `domainname` -h NISMASTER -c prpasswd
```

```
# /var/yp/ypxfr -d `domainname` -h NISMASTER -c prpasswd_nonsecure
```

In the example, substitute the name of the local NIS master server for *NISMASTER*. This will transfer initial copies of those maps for those slave servers.

- A login process that encounters a login failure has to check the `prpasswd` map for the latest unsuccessful login information. This requires an up-to-date `prpasswd` map. Thus, the `yppush` operation for the `prpasswd` map must occur for each failed login; that map (at least) must be pushed during the normal operation of the `rpc.yppasswdd` daemon. Setting the `/var/yp/Makefile` variable `NOPUSH` is not recommended for such configurations.
- Sites that cannot use NIS to share `prpasswd` information may want to use NFS to share the `/tcb/files` and `/var/tcb/files` directories. This requires exporting the directories with root access to the participating nodes with `-root=client1:client2:client3` or `-root=0`, as appropriate. See the `exports`(4) reference page. It also requires that NFS locking be enabled so that database corruption does not occur.

9.5.7 Troubleshooting NIS

Table 9–2 discusses some common NIS problems and possible reasons for those problems.

Table 9–2: NIS Troubleshooting

Problem	Possible Reason
Successful login to a local account, but cannot log in to any of the NIS accounts. The `dxaccounts` utility displays that the account exists and is not locked.	1. Check the `/etc/svc.conf` file and see if it contains the line `auth=local,yp`. 2. Check the `/etc/passwd` file and see if there is a "`+:`" as the last line of the file.

Slave NIS server does not get the updated `prpasswd` maps on boot.	Check the `/var/yp/ypxfr_1per-day`, `/var/yp/ypxfr_1perhour`, and `/var/yp/ypxfr_2perday` files and verify that each contains the lines:
	`ypxfr -a "$method" prpasswd`
	`ypxfr -a "$method" prpasswd_nonsecure`

Table 9–2: NIS Troubleshooting (cont.)

Problem	Possible Reason
The `dxaccounts` program View popup menu does not show any NIS User Account Database options (for example, NIS Users, NIS Groups, and NIS Templates).	NIS is not running or has not been configured.
When you issue the `make` command from `/var/yp`, you get the message `Map 'ypslaves ' is empty for domain 'domainname'`	This is an informational message. No action is required.

When you issue the `make` command from `/var/yp`, you get the message `Map 'hosts.byname' is empty for domain 'domainname' cant bind to master for domainname hosts.byname no such map in server's domain will use slave copy!`	The hosts map does not exist. Perform the following commands:
	`# touch /var/yp/src/hosts`
	`# cd /var/yp`
	`# make`

10

Administering the Audit Subsystem

This chapter describes the purpose of system auditing, how auditing is performed, what activities should be audited, and how to read and respond to audit reports. Responsibilities, managing events, tools, and generating reports are also described.

10.1 Overview of Auditing

In its simplest form, auditing consists of the following:

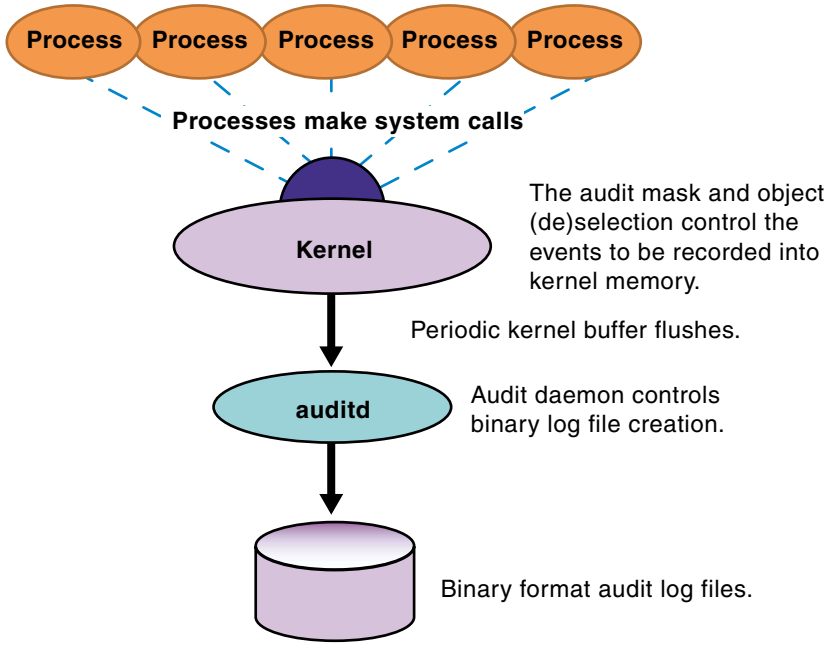
1. An activity on the system results in the generation of an audit record. This audit record contains information about the activity, such as what the activity was, when it occurred, and the ID of the user who caused it.

   As security manager, you decide which activities will result in the generation of audit records. This choosing of what to audit is sometimes called preselection.

2. The audit record is stored along with other audit records in a file, the audit log.
3. You use a utility to select information from the audit log file and to generate reports from that information. This process is sometimes called post-reduction.

Figure 10–1 provides an overview of the audit subsystem.

Figure 10–1: The Audit Subsystem



Auditing provides you with a powerful tool for monitoring activity on the system. Through auditing, you can accomplish the following:

- Discourage users from attempting to violate security. A user who knows that system activities are monitored and that security violations can be tracked to the responsible individual might be dissuaded from attempting to violate security.
- Detect attempts at violations or activities indicative of probing a system for weak points. If an audit reveals failed attempts to violate system security, you can take counter measures to lessen the likelihood of later attempts succeeding.
- Assess damage and restore the system if a break-in should occur. Careful analysis of an audit trail after a break-in can help you determine what occurred during the security violation and what steps are needed to return the system to its original state. It also allows you to take steps to prevent similar break-ins in the future.
- Evaluating and debugging application software. The capability of audit to monitor the completion status and arguments of system calls for a designated process provides the ability to assess what is going on inside an application. You can also determine which files the application is

attempting to access even if you do not have the source code. Refer to Section 10.14 for more information.

It is important that you inform users of the purpose and, in general terms, the nature of the auditing performed on the system. Present auditing in a positive light, as a tool to help protect the users' files and their access to system resources. This helps minimize any resentment; users who are openly told that their system is regularly audited are less likely to feel as though they are being spied upon. For those users who might be tempted to violate security, knowledge that activities are monitored can be a powerful deterrent.

To manage the audit subsystem, you perform the following tasks:

- Configure the audit system
- Select activities to be audited
- Produce audit reports
- Manage disk space used by the audit subsystem.
- Archive audit files

10.1.1 Audit Files

Table 10–1 describes the files used by the audit subsystem.

Table 10–1: Files Used for Auditing

File Name	Security-Relevant Information
/var/audit/auditlog.*hostname.nnn*	Default log file. *hostname* is the name of the system that generated the audit log. *nnn* is a generation number between 000 and 999.
/var/adm/syslog.dated\ /current/daemon.log	Default log file for status messages from the audit subsystem.
/etc/sec/audit_events	A file listing all the system activities that can have security relevance. This file can be used as input to the auditmask command, which controls which events are audited on the system.
/etc/sec/site_events	A file defining site-specific audit events. It supports integrating application auditing into the audit subsystem.
/etc/sec/event_aliases	A file containing a list of aliases that represent sets of events that can be audited.

Table 10–1: Files Used for Auditing (cont.)

File Name	Security-Relevant Information
/etc/sec/auditmask_style	A file defining audit style flags for profiles.
/etc/sec/file_objects/*	Directory containing lists of filenames to monitor for profiles.
/etc/sec/rc_audit_events	A file containing the list of audit events to monitor. The list is loaded at system startup, and the file is pointed to by AUDITMASK_FLAG in /etc/rc.config or /etc/rc.config.common.
/etc/sec/fs_objects	A list of files that are subject to audit selection or deselection.
/etc/sec/auditd_loc	A list of alternate paths and hosts where audit logs can be stored if the current location becomes unavailable or full.
/etc/sec/auditd_clients	A list of the remote hosts that can send their audit data to be stored on the local system.
/cluster/members/{*memb*}/dev/\audit	A CDSL required for audit on clustered systems.
/cluster/members/{*memb*}/dev\.audit/audS	A CDSL required for audit on clustered systems.

The Tru64 UNIX audit subsystem records activities in any file at any location chosen by the system administrator; the default log file is /var/audit/auditlog.*hostname*.*nnn*.

10.1.2 Audit Tools

The tools for auditing on Tru64 UNIX systems can be divided into two categories:

- The audit subsystem, which has powerful features unique to the trusted Tru64 UNIX operating system. This is the trusted method for performing security-relevant auditing.
- Traditional UNIX operating system logging features, such as the system accounting files and the last command.

The audit subsystem provides a choice of the events to be logged, flexible data reduction of the audit log, and ease of maintenance. These features can be accessed through either a powerful command-line interface or through an easy-to-use graphic interface that provides point-and-click operation and on-line help.

10.1.2.1 Command-Line Interface

The following commands are used with the audit subsystem:

`sysman auditconfig`	Establishes the audit environment on your system.
`auditmask`	Selects events for inclusion in the audit log or displays a list of events currently being recorded in the audit log.
`audgen`	Provides the ability, from the command line, to generate a log record containing a message of your choice.
`auditd`	Activates the auditing daemon (turns on auditing), administers audit data storage, and configures the audit subsystem.
`audit_tool`	Selectively extracts information from the audit log and presents it in a readable form.
`audit_tool.ultrix`	Selectively extracts information from an audit log created on an ULTRIX system and presents it in a readable form.

Use of these commands is limited to those with superuser status.

10.1.2.2 Graphical Interface

The graphical interface to the audit subsystem is accessed using the CDE Dashboard as follows:

System Applications→Daily Administration→Audit Manager

Configuration of the audit subsystem is done with the `sysman auditconfig` command. All other activities can be handled with the graphical interface. If you intend to use the graphical interface rather than the command-line interface, you will want to read Section 10.4.1 for an overview of the `auditd` command, but you can skip the other command-line information in Section 10.4.

Use of the graphic interface is limited to those with superuser status.

10.2 Basic Audit Configuration

Use the procedures in this section if you want a simple, default audit configuration.

The audit subsystem configuration is done in two parts. The first part configures the kernel for auditing and establishes audit log management parameters. You select the location for the audit logs, the action that audit takes if file space is exhausted, and for how long the audit logs are held (forever if specified) on your system before being deleted to free disk space. In the second part, you select the events you want to audit.

Note

Auditing must be built into your kernel. In addition to system-wide auditing, you may specify auditing of events on a user-by-user basis by including enhanced security subset OSFC2SEC510. The `dxaudit` graphical interface for audit requires the software subset OSFXC2SEC510.

You can check to see if these subsets have been installed as follows:

```
# setld -i | grep "C2SEC5.."
OSFC2SEC510    installed    Enhanced Security (System Administration)
OSFXC2SEC510   installed    Enhanced Security GUI (System
Administration)
```

If the subsets do not show as installed, refer to the `setld`(8) reference page for information on how to install these subsets.

The following steps show you how to quickly get the audit subsystem running in a simple, default configuration. The screen names that are referenced are found in the title bar of each screen:

1. Begin by executing the `sysman auditconfig` command to start configuring audit. Note that if the audit subsystem is not configured into your system's kernel, `sysman auditconfig` guides you through a kernel rebuild.

   a. Click Yes on the Welcome screen to begin configuring audit.

   b. Click OK on the Information screen.

   c. Accept the default for the audit log location (`/var/audit/audit-log.`*`hostname`*`.`*`nnn`*) by clicking Next on the Log Pathname screen.

   d. Accept the default for the action if log space is exhausted (Suspend auditing until space becomes available) by clicking Next on the Action On Log File Space Exhaustion screen.

e. Accept the default for the audit log lifespan in months (forever), and the hour of deletion (3) on the Log File Lifespan screen.

f. Accept the default for audit console message destination (`syslog-/var/adm/syslog.dated/current/daemon.log`) by clicking Finish Part One on the Advanced Audit Options screen.

g. Proceed to part two by clicking Yes to the question Do you wish to proceed to part two? on the Audit Event Information screen.

2. In part two of configuring audit, you choose your auditing profile or category. Profiles and categories are capitalized in the menu list; there are six to choose from:

Profile/Category	Description
Desktop	Suggested minimal auditing for a single user system.
NIS_server	Suggested auditing for a system used as a NIS server.
Networked_system	Suggested auditing for a system on a network.
Server	Suggested auditing for a system that is used as a server for network-based applications.
Timesharing	Suggested minimal auditing for a system that is used to support multiple interactive users.
Timesharing_extended_audit	Extended auditing for a system that is used to support multiple interactive users.

a. Select the profile or category closest to the configuration of your system and click Next on the Audit Event Category Selection screen.

Note that use of the CTRL key allows the selection of more than one category; for example, `Desktop` and `Networked_system` for a desktop system on a network. While multiple selections are possible, use the minimum profile that meets your needs.

Accept the default list of events to be audited by clicking Next on the Advanced User Audit Event Modification/Deletion screen.

b. Some profiles and categories include this step, otherwise skip to the next step. Accept the default list of files to be audited by clicking Next on the UNIX File System Objects screen. Note that this menu item does not appear for all profiles and categories.

c. Accept the options for audit events that have been set as a result of the profile or category that were selected by clicking Finish on the Advanced Options screen.

3. Complete the audit set up by clicking OK on the Audit Configuration Complete screen.

4. Check the configuration of the audit subsystem by entering the `auditd -w` command. If you took the system default, your configuration looks like the following:

```
# auditd -w

Audit data and msgs:
-l) audit data destination                = /var/audit/auditlog.hostname.001 [1]
-c) audit console messages                = syslog [2]

Network:
-s) network audit server status (toggle)  = off [3]
-t) connection timeout value (sec)        = 4

Overflow control:
-f) % free space before overflow condition = 10 [4]
-o) action to take on overflow            = suspend audit [5]
```

[1] The name of the audit log.

[2] The location of audit subsystem messages. Changes in the status of the audit subsystem are recorded here.

[3] Auditing across the network is not enabled (remote clients may not log to this system).

[4] Percent of remaining free space in the file system that will trigger an audit overflow condition. In this case, 10 percent.

If the file system containing the audit log becomes 90 percent full, the audit subsystem takes an overflow action.

[5] The overflow action is to suspend auditing until storage space is available.

In place of *alternate file systems*, any directory names specified in `/etc/sec/auditd_loc` are displayed.

10.3 Advanced Configuration of Audit

This section provides audit procedures to establish a customized audit configuration. Advanced users will find pointers to more detailed information about the configuration questions. The responses supplied in this section are suggestions; once you become familiar with the audit configuration process, you may want to select different responses.

Note

The auditmask features that allow you to specify auditing of events on a user-by-user basis require the enhanced security subset OSFC2SEC510. The `dxaudit` graphical interface for audit requires the software subset OSFXC2SEC510.

You can check to see if these subsets have been installed as follows:

```
# setld -i | grep "C2SEC5.."
OSFC2SEC510    installed    Enhanced Security (System Administration)
OSFXC2SEC510   installed    Enhanced Security GUI (System
Administration)
```

If the subsets do not show as installed, refer to the `setld`(8) reference page for information on how to install these subsets.

The audit subsystem configuration is done in two parts. The first part configures the audit logs your system creates. You select the location for the audit logs, the action that audit takes if file space is exhausted, and how long the audit logs are held (forever if specified) on your system before being deleted to free disk space. In the second part, you select the events you want to audit.

The following steps show you how to get the audit subsystem up and runing in a custom configuration for your site. The screen names that are referenced are found in the title bar of each screen:

1. In part one of configuring audit, you execute the `sysman auditconfig` command to start configuring audit.

   a. Click on Yes on the Welcome screen to begin configuring audit.

   b. Click on OK on the Information screen.

   c. Select the audit log location (`/var/audit/auditlog.`*`host-name.nnn`*) by selecting Next on the Log Pathname screen.

   d. Accept the default for the action if log space is exhausted (Suspend auditing until space becomes available) by selecting Next on the Action On Log File Space Exhaustion screen. Refer to `-o` option in the `auditd`(8) reference page for more information.

   e. Select the audit log lifespan in months (forever), and the hour of deletion (3) on the Log File Lifespan screen. Refer to Section 10.6.2.1.

   f. Select the audit console message destination (`syslog` — `/var/adm/syslog.dated/current/daemon.log`) by selecting Finish Part One on the Advanced Audit Options screen. Refer to Section 10.8.

g. Proceed to part two by selecting Yes to the question Do you wish to proceed to part two? on the Audit Event Information screen.

2. In part two of configuring audit, you need to pick from a list of six profiles or categories. The profiles set up audit style (`/etc/rc.config.common` AUDITMASK_FLAG), audit events (`/etc/sec/rc_audit_ events`), and files to be audited (`/etc/sec/fs_objects`), which are used to set up the audit subsystem at system startup. The first letter of the profiles are capitalized in the menu list; there are six to choose from:

Profile/Category	Description
Desktop	Suggested minimal auditing for a single user system.
NIS_server	Suggested auditing for a system used as a NIS server.
Networked_system	Suggested auditing for a system on a network.
Server	Suggested auditing for a system that is used as a server for network-based applications.
Timesharing	Suggested minimal auditing for a system that is used to support multiple interactive users.
Timesharing_extended_audit	Extended auditing for a system that is used to support multiple interactive users.

   a. Select the profile or category closest to the configuration of your system and click Next on the Audit Event Category Selection screen. Refer to Section 10.6.1.6.

   b. Select the default list of events to be audited by clicking on Next on the Advanced User Audit Event Modification/Deletion screen. Refer to Section 10.6.1.

   c. Select the default list of files for auditing by clicking on Next on the UNIX File System Objects screen. Note that this menu item does not appear for all profiles and categories. Refer to Section 10.6.1.5.

   d. Select the options for audit events that have been set as a result of the profile or category that was selected by clicking Finish on the Advanced Options screen. Refer to Section 10.6.1 and the `-s` option in the `auditmask`(8) reference page.

3. Complete the auditing setup by clicking OK on the Audit Configuration Complete screen. Audit is now running.

4. Check the configuration of the audit subsystem by entering the `auditd -w` command. If you took the system default, your configuration looks like the following:

```
# auditd -w

Audit data and msgs:
-l) audit data destination                = /var/audit/auditlog.hostname.001 [1]
-c) audit console messages                = syslog [2]

Network:
-s) network audit server status (toggle)  = off [3]
-t) connection timeout value (sec)        = 4

Overflow control:
-f) % free space before overflow condition = 10 [4]
-o) action to take on overflow            = suspend audit [5]
```

[1] The name of the audit log.

[2] The location of audit subsystem messages. Changes in the status of the audit subsystem are recorded here.

[3] Auditing across the network is not enabled (remote clients may not log to this system).

[4] Percent of remaining free space in the file system that will trigger an audit overflow condition. In this case, 10 percent.

If the file system containing the audit log becomes 90 percent full, the audit subsystem takes an overflow action.

[5] The overflow action is to suspend auditing until storage space is available.

10.4 Audit Commands

The `auditd`, `auditmask` and `audit_tool` commands are described in this section. The `auditd` and `auditmask` commands control the audit subsystem. The `audit_tool` command generates reports from the output of the audit subsystem.

Changes made to the audit subsystem with `auditd` and `auditmask` commands are temporary and are reset on a system reboot.

The current system defaults for the audit subsystem are stored in the `/etc/rc.config.common` file.

AUDITMASK_FLAG is used to pass command-line arguments to `auditmask` at system startup. AUDITMASK_FLAG sets the audit style flags and sets `auditmask` to read events selected for auditing from `/etc/sec/rc_audit_events`. The `/etc/sec/fs_objects` file contains a list of files that have modified property lists (modified during audit subsystem configuration) that allow the files to be monitored on object

selection/deselection. Refer to the auditmask(8) reference page for more information on object selection/deselection.

10.4.1 Configuring the Audit Subsystem: the auditd Command

The auditd command turns on and off the audit daemon and is used to configure the audit daemon. Table 10–2 gives examples of uses of the auditd command. For a complete description, see the reference page for auditd(8).

Table 10–2: auditd Examples

Configuration Task	Command
Turn auditing on	auditd
Turn auditing off	auditd -k
Designate location of audit log	auditd -l *pathname*
Flush the kernel audit buffer	auditd -d
Configure the audit subsystem so that if a situation should arise where the current log cannot be written to, a new log will be started in a different location (first, you will need to put a list of alternate pathnames in the file /etc/sec/auditd_loc)	auditd -o changeloc
Suspend auditing if the audit log cannot be written to	auditd -o suspend
Halt the system if the audit log cannot be written to	auditd -o halt
Enable the local system to accept audit data from foreign hosts (first you must put a list of permitted hosts in /etc/sec/auditd_clients)	auditd -s (This command toggles network auditing. If network auditing is on, auditd -s turns it off. If it's off, auditd -s turns it on.)

10.4.2 Selecting Events to Audit: The auditmask Command

The auditmask command sets the system and process audit masks, which determine what gets audited on the system. The following table gives examples of uses of the auditmask command. For a complete description, see the reference page for auditmask(8).

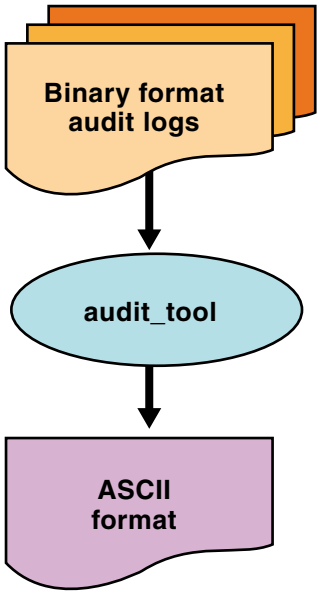
Selection Task	Command
Display list of currently audited events and style flag settings	auditmask

Selection Task	Command
Audit all events. This option causes a significant degradation in system performance.	`auditmask -f`
Do not audit any events.	`auditmask -n`
Audit successes and failures of an event.	`auditmask` *`eventname`* or `auditmask` *`eventname`*`:1:1`
Audit only successes of an event.	`auditmask` *`eventname`*`:1:0`
Audit only failures of an event.	`auditmask` *`eventname`*`:0:1`
Turn off auditing of an event. Once an event is designated for auditing, auditing of the event continues until explicitly turned off.	`auditmask` *`eventname`*`:0:0`
Audit events as specified in a file. See `/etc/sec/audit_events` for an example of such a file.	`auditmask <` *`filename`* or `auditmask < /etc/sec/rc_audit_events`
Audit all processes generated by a specific user.	`auditmask -a` *`AUID-of-user`*
Audit file activity only for certain files (Object Selection).	`auditmask -x` *`filename`*`[:1|:0]` or `auditmask -X /etc/sec/fs_objects[:1|:0]` See Section 10.6.1.5 for details.
Do not audit file activity for certain files (Object Deselection).	`auditmask -y` *`filename`* or `auditmask -Y` *`filelist`* See Section 10.6.1.5 for details.

10.4.3 Producing Audit Reports: The `audit_tool` Command

Figure 10–2 illustrates how `audit_tool` converts binary data to an ASCII report. Table 10–3 gives examples of how to use the `audit_tool` command. For a complete description, see the reference page for `audit_tool`(8).

Figure 10–2: Audit Report Formats



Contains all records generated by the combination of the audit mask and, if enabled, file object (de)selection.
Flush the kernel buffer if needed using the audit -d command.

The audit_tool utility selects requested records from the binary log files and generates ASCII output. Events, dates, audit ID, PID and so on can be used as selection criteria.

ZK-1581U-AI

Table 10–3: Generating Audit Reports with the `audit_tool` Command

Report Task	Command
Generate a report for all activities that occurred during a certain time period.	`audit_tool -t` *start* `-T` *end* *auditlog* The *start* and *end* times are in the format `yymmdd[hh[mm[ss]]]`. Hours (`hh`), minutes (`mm`), and seconds (`ss`) are optional.
Generate a report for all failed attempts by a specific user to open files.	`audit_tool -U` *username* `-e open:0:1` *auditlog* If `login` is not preselected for auditing, then no user names appear in the audit log. In this case, you can still select for UIDs (`audit_tool -u` *UID*) or AUIDs (`audit_tool -a` *AUID*).
Generate a report for each audit ID (AUID) in the audit log. The reports will be named `report.`*AUID*.	`audit_tool -R` *auditlog*
Generate a report of all audit records containing the text *string* in a parameter field or associated with a descriptor.	`audit_tool -s` *string*

Table 10–3: Generating Audit Reports with the `audit_tool` Command (cont.)

Report Task	Command
Generate a report of all audit records associated with a specific process ID.	`audit_tool -p` *PID*
Generate a report of all audit records associated with a specific process ID and all the descendants of that process.	`audit_tool -p -`*PID*

10.5 What to Audit

There are three categories of auditable events on a Tru64 UNIX system:

- System calls (including Mach events)
- Trusted events
- Site-defined events

All system calls and trusted events of possible security relevance are listed in the file `/etc/sec/audit_events`.

To generate a list of all possible auditable events, including site-defined events, and save it in the file `all_auditable_events`, enter the following commands:

```
# auditmask > original_audit_mask  [1]
# auditmask -f  [2]
# auditmask > all_auditable_events  [3]
# auditmask -n  [4]
# auditmask < original_audit_mask  [5]
```

Note

The `auditmask -f` command may slow down your system.

[1] Save the current audit mask to the file `original_audit_mask`.

[2] Set the audit mask to all events.

[3] Save the audit mask to the file `all_auditable_events`.

[4] Set the audit mask to no events.

[5] Restore the audit mask to the original events.

10.5.1 Trusted Events

A trusted event is an event that is associated with a security protection mechanism; it does not always correspond directly to a system call. A list of the trusted events follows:

`audit_daemon_exit`	Indicates that the audit daemon exited abnormally. This occurs only when there is insufficient memory available during initialization of the audit daemon. The exit is recorded in the new audit log, and a message is displayed on the designated audit console.
`audit_log_change`	Indicates that the audit daemon closed the current audit log and began writing a new log (for example, in response to the `auditd -x` command). The change in logs is recorded in the current audit log, and a message is displayed on the designated audit console.
`audit_log_create`	Indicates that a new audit log was created in response to the removal of the current log file. The new file has the generation number of the lost log file incremented by 1. The creation of the new log is recorded at the beginning of the new audit log, and a message is displayed on the designated audit console.
`audit_log_overwrite`	Indicates that the audit daemon began overwriting the current audit log as you specified with the `-o overwrite` option to `auditd`. The overwrite is recorded at the beginning of the newly overwritten audit log, and a message is displayed on the designated audit console.
`audit_reboot`	Indicates that the audit daemon initiated a system reboot (as a result of an overflow of the log) as you specified with the `-o halt` option to `auditd`. The reboot is recorded at the end of the current audit log, and a message is displayed on

	the designated audit console before the reboot occurs.
auditconfig	Indicates that the –o changeloc option to the auditd command was used to change the specified overflow action. The change in the audit setup is recorded in the current audit log.
audit_start	Indicates that the audit daemon has been started.
audit_stop	Indicates that the audit daemon was killed normally (typically, with the –k option to auditd). The shutdown is recorded at the end of the current audit log, and a message is displayed on the designated audit console when the shutdown occurs.
audit_suspend	Indicates that the audit daemon suspended auditing (as a result of an overflow of the log) as you specified with the –o suspend option to auditd. The suspension is recorded in the current audit log, and a message is displayed on the designated audit console.
audit_xmit_fail	Indicates that the audit daemon was sending audit records across a network and the transmission failed. The failure is recorded in the local log specified as the next path in /etc/sec/auditd_loc (with auditd –r) or the default local path (/var/adm).
audgen8	The audgen8 command (a command-line interface to the audgen() routine) was used to generate an audit record.
auth_event	An event associated with user-authentication and the management of user accounts occurred. Trusted

	auth_events include passwd, su, rsh, and login. The event is recorded in the current audit log.
login	A user attempted to log in to the system.
logout	A user logged out of the system.

10.5.2 Site-Defined Audit Events

A site can define its own audit events (referred to as site-defined events). This is useful if you want applications to generate records specific to their activities.

Trusted application software can generate data for the site-defined events and subevents. The data can be included in the audit logs with the system's audit data or stored in application-specific logs.

Both preselection and postreduction capabilities are supported for site events. That is, you can use the auditmask and audit_tool commands on site-defined events exactly as you do for other audit events. Postreduction capabilities are also supported for subevents.

The system administrator must create an /etc/sec/site_events file, which contains the event names and event numbers for the system's site events. The site_events file has one entry for each site event. Each site event entry may contain any number of subevents.

The lowest allowed site event number is MIN_SITE_EVENT, which is defined in <sys/audit.h>. Typically, the number is 2048. By default, up to 64 site events can be defined. However, this upper limit can be increased up to a maximum 1,048,576.

To change the upper limit for the allowed number of site events, add an entry in the /etc/sysconfigtab file. For example, to allow up to 5000 site-defined events, use the sysconfigdb command to add the following lines to /etc/sysconfigtab:

```
sec:
    audit-site-events=5000
```

Then reboot the system.

Once /etc/sec/site_events has been set up, applications can use the audgenl() library routine to generate application-specific audit data.

Programming information about providing application-specific auditing is found in the reference page for audgenl(3).

10.5.3 Dependencies Among Audit Events

Some information in the audit log is based on previously audited events. For example, the LOGIN event associates a login name with a real UID (RUID). Subsequent occurrences of that RUID (for a given process) can then be associated with a login name. Such data is called state-dependent information. The following three audit records illustrate state-dependent information. The first record shows a successful `open()` of `/etc/passwd`, returning a value of 3:

```
audit_id:    1621      ruid/euid: 0/0         (username: root)
pid:         23213     ppid: 23203            cttydev: (6,1)
procname:    state_data_test
event:       open
char param:  /etc/passwd
flags:       2 : rdwr
vnode id:    2323       vnode dev:   (8,1024)   [regular file]
object mode: 0644
result:      3 (0x3)
ip address:  16.153.127.241 (alpha1.sales.dec.com)
timestamp:   Wed Nov 10 17:49:59.93 2000
```

The following record shows the result of an `ftruncate()` system call for the `/etc/passwd` file with state-dependent information. The state-dependent data currently associates the file name `/etc/passwd` with descriptor 3 for this process:

```
audit_id:    1621      ruid/euid: 0/0         (username: root)
pid:         23213     ppid: 23203            cttydev: (6,1)
procname:    state_data_test
event:       ftruncate
vnode id:    2323      vnode dev:   (8,1024)   [regular file]
object mode: 0644
descriptor:  /etc/passwd (3)
result:      0
ip address:  16.153.127.241 (alpha1.sales.dec.com)
timestamp:   Wed Nov 10 17:49:59.96 2000
```

If state-dependent data is not being maintained, you would see only that the `ftruncate()` system call was against descriptor 3 (vnode ID = 2323, dev = 8,1024):

```
audit_id:    1621      ruid/euid: 0/0         (username: root)
pid:         23213     ppid: 23203            cttydev: (6,1)
event:       ftruncate
vnode id:    2323      vnode dev:   (8,1024)   [regular file]
object mode: 0644
descriptor:  3
result:      0
ip address:  16.153.127.241 (alpha1.sales.dec.com)
timestamp:   Wed Nov 10 17:49:59.96 2000
```

Table 10–4 lists state-dependent information and the audit events required to maintain it.

Table 10–4: State-Dependent Information

To get this state-dependent Information...	...audit these events
login user name	`login`
process name	`execv, execve, exec_with_loader`
file name	`open, socket, bind, dup, dup2, fcntl`
current directory	`chdir, chroot`

The `exit()` system call informs `audit_tool` that it no longer needs the state-dependent information for the exiting process. This allows `audit_tool` to run faster.

If you are not interested in state-dependent data, you do not need to audit `exit()`, and use the `–F` option to the `audit_tool` program for fast mode.

See the `audit_tool(8)` reference page for more information.

10.6 Managing the Volume of Audit Data

The audit subsystem is capable of generating and collecting large amounts of data. Therefore, it is important for the system administrator to place reasonable limits on the data being generated by the audit subsystem and to monitor the growth of the audit log data stored on system. This monitoring is needed to make sure that audit logs stored online do not grow to fill the file system and cause a denial of service situation.

10.6.1 Before the Audit Data Is Collected

The audit subsystem provides the following features that allow the system administrator to preselect events for auditing and to easily change those preselections:

- Audit masks and control flags
- Event aliases
- Object selection and object deselection

10.6.1.1 Audit Masks

Audit masks determine which events are audited. There are two kinds of audit masks:

system audit mask — Applies to all processes on the system.

process audit mask — Applies to a specific process.

A special case of the process mask is the login process mask. The login process mask for a user is stored in the password database (`/tcb/files/auth.db`) in the `u_auditmask` field.

When the auditmask is set for a user login process, at user login, that audit mask is applied to the login process, and all offspring processes inherit that auditmask.

The per-user auditmask feature, which allows you to specify auditing of events on a user-by-user basis, requires that enhanced security be configured and active.

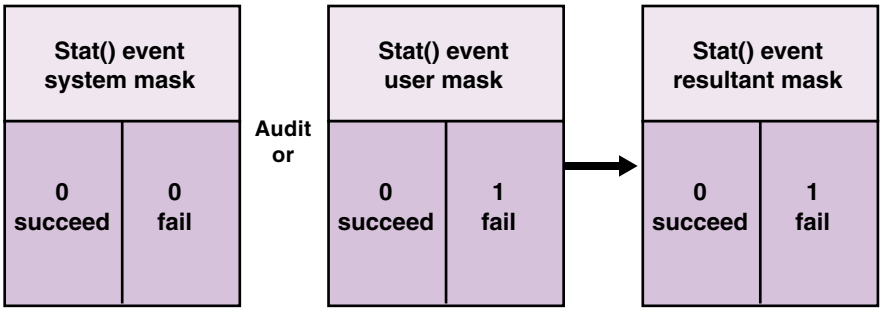
Interaction between system audit mask and process mask	**Control flag value for `auditmask -c`**
Generate audit record for an event if either the system mask or process mask specifies the event.	`auditmask -c or`
Generate audit record for an event only if both the system mask and process mask designate the event.	`auditmask -c and`
No auditing for this process.	`auditmask -c off`
Generate audit records based only on the process audit mask.	`auditmask -c usr`

10.6.1.2 Audit Control Flags

The audit control flag mediates the interaction between the system audit mask and the process audit mask. Figure 10–3 illustrates how the system and process masks interact.

Figure 10–3: System and Process Audit Mask Interaction

Enhanced security adds the capability of per user audit masks which can be used in conjunction with the system audit mask. Logical operations are used to derive the result of the masks. The default is audit_or. User audit masks are stored in the trusted password database and are most easily added by dxaudit.



The audit control flag regulating system mask and process mask interaction can be set from the `dxaudit` program found on the CDE Dashboard, as follows:

Application Manager→Daily Administration→Audit Manager→ Collection→ Modify Active Process Mask

10.6.1.3 User Process Control Flag

In the case of a user's login-process mask, the `u_auditcntl` field in the user's entry in the enhanced (protected) password database stores the value for the control flag.

The control flag for a user's login process is set with the `dxaccounts` program found on the CDE Dashboard, as follows:

Application Manager→Daily Administration→Account Manager→Security Attributes→Login Process Mask

10.6.1.4 Event Aliases

An event alias groups multiple audit events under a single name. You define an event alias in the file named `/etc/sec/event_aliases`. The file consists of a series of alias definitions. Each definition has the following format:

[*alias: event[:success:failure] [event[:success:failure] ...]*]

An event can be a system call, trusted event, site event, or another alias. The success and fail notation is the same one used for `auditmask` and `audit_tool`. Continuation lines are allowed.

A few carefully defined event aliases can provide a wide range of auditing coverage, allowing you to easily change auditing preselection.

Event aliases are selected for auditing just as any other event — with the `auditmask` command or the Audit Manager graphical interface.

See `/etc/sec/event_aliases` for an example of an alias file. The Tru64 UNIX event aliases and profiles are defined in this file. Compaq reccommends that you add your own aliases to the end of the `/etc/sec/event_aliases` file.

10.6.1.5 Object Selection and Deselection

The object selection and deselection modes provide another preselection mechanism designed to help administrators audit specifically those operations of interest to them. The file object selection and deselection mechanism provides a further level of granularity for events which operate on files.

Events such as `mount` and `reboot` affect system state. Data access events, such as `open` and `stat`, act on files. Although `reboot` attempts might always be security relevant, not all file `open` events are (depending on the site security model).

Object selection and deselection let you chose which files do (selection) or do not (deselection) result in audit records when those files are objects of data access operations.

The data access operations that can be selected or deselected are:

```
read        open              close
link        lseek             access
stat        lstat             dup
open        revoke            readlink
fstat       pre_F64_stat      pre_F64_lstat
dup2        pre_F64_fstat     readv
pread       getdirentries     stat
lstat       fstat             _F64_readv
```

Object selection and deselection work as follows:

Selection
: The object selection mode provides the ability to specify a set of files for which selected events are audited, while those same events on other files are not audited. In the selection mode, audit records are generated only when an event is selected and

either that event is acting on a selected file or not acting on any file.

For example, you can flag the `/etc/passwd` and `/.rhosts` files, enable object selection, and audit the `open` system call. This causes an `open` on `/etc/passwd` and `/.rhosts` to be audited while an open on `/tmp/`*xxxx* files is not audited.

Deselection

The file deselection mode provides the ability to specify a set of files for which specific selected events are not audited, while those same events on other files are audited. The events that may be deselected are data access operations (no data modifications). File `open`'s for write or truncate access are examples of file modification.

In the deselection mode, audit records are generated for selected events, unless all files operated on by the system call are deselected and the operation is a data access. So, if you are auditing `stat` and `unlink`, and the file `foo` is deselected, then a `stat` of `foo` would not be audited, but an `unlink` of `foo` would be audited (the `unlink` is not a data access operation). The result is that it is now possible, for example, to not audit accesses to the `/usr/shlib/libc.so` library, but still audit `open`'s of the `/etc/passwd` file.

Audit selection and deselection do not reduce the auditing of processes for which `auditmask -c usr` was specified (the value of the audit control flag is `AUDIT_USR`).

Using object selection and deselection requires the following three steps:

1. Decide which file or files you want to apply object selection or deselection to. If you want to apply selection or deselection to more than one file, you can create a file that is a list of file names (complete pathnames), one file name per line.
2. Set the audit style either to Object Selection or Object Deselection as appropriate:

   From Audit Manager: Collection→Audit Style

   From the command line: `auditmask -s obj_sel` or `auditmask -s obj_desel`
3. Activate object selection and deselection as follows:

   Object selection for one file:

   ```
   # auditmask -x filename
   ```

Object selection for a series of files listed in a file named *filelist*:

```
# auditmask -X filelist
```

Object deselection for one file:

```
# auditmask -y filename
```

Object deselection for a series of files listed in a file named *filelist*:

```
# auditmask -Y filelist
```

The following are examples of how to enable the selection of a file for audit:

```
# auditmask –s obj_sel
# auditmask –q /etc/passwd
selection: off     deselection: off  --  /etc/passwd
# auditmask –x /etc/passwd
selection: off => on   --  /etc/passwd
# auditmask –q /etc/passwd
selection: on     deselection: off  --  /etc/passwd
```

The following example shows how to disable the selection of a file for audit:

```
#  auditmask -x /etc/passwd:0
```

The following example shows how to deselect a list of files:

```
# auditmask –s obj_desel
# cat desel_file
/etc/motd
/etc/fstab
/etc/passwd
# auditmask –Q desel_file
selection: off    deselection: off  --  /etc/motd
selection: off    deselection: off  --  /etc/fstab
selection: off    deselection: off  --  /etc/passwd
# auditmask –Y desel_file
deselection: off => on   --  /etc/motd
deselection: off => on   --  /etc/fstab
deselection: off => on   --  /etc/passwd
```

The status of the object selection and deselection flags for a file list can be displayed from the CDE dashboard as follows:

1. Select Application Manager→Daily Administration→Audit Manager→Modify System Mask→Current→Edit →Object Selection/Deselection.
2. Under File, select `/etc/sec/fs_objects`.

3. Under File, select List of Files.
4. Settings are not applicable.
5. Click on Query.

Although the system object selection and object deselection audit styles are mutually exclusive, it is possible for any one object to be subject to both models simultaneously on different systems (across NFS). The `proplistd` daemon needs to be running to transfer attributes across NFS.

10.6.1.6 Audit Profiles and Categories

Audit profile and categories provide a method to consolidate the parameters that define what is monitored by the audit subsystem and to group that information under a profile. For this section the terms profile and categories are considered synonymous. Profiles are made up of of three parts:

Audit style information	Located in `/etc/sec/auditmask_style` under the `Profile:` label. Refer to the reference page `auditmask`(8) under the `-c` option for a description of the valid audit style characteristics.
Audit events to be monitored	Located in `/etc/sec/event_aliases` under the `Profile:` label.
Files to be monitored	The list of files is in the `/etc/sec/file_objects/Profile` file. Monitoring is performed by setting the appropriate object selection and deselection flags. Information in `/etc/sec/auditmask_style` determines which flags are set. This file is optional and does not need to exist if the audit style defined in `/etc/sec/auditmask_style` does not include `obj_sel` or `obj_desel`.

The following profiles are defined:

Profile	Description
Desktop	Suggested minimal auditing for a single user system.
NIS_server	Suggested auditing for a system used as a NIS server.
Networked_system	Suggested auditing for a system on a network.
Server	Suggested auditing for a system that is used as a server for network based applications.

Profile	Description
Timesharing	Suggested minimal auditing for a system that is used to support multiple interactive users.
Timesharing_extended_audit	Extended auditing for a system that is used to support multiple interactive users.

Note

You should audit the `logout()` event. This makes the `audit_tool` program run faster when reducing audit data.

During phase two of the audit configuration, the `auditconfig` utility takes the information from the user-specified `Profile`, modifications to the audit style information, and the events to be audited list (the files to monitor list cannot be changed by the `auditconfig`). When you have completed the selections, the `auditconfig` does the following:

- Modifies AUDITMASK_FLAG in `/etc/rc.config.common`
- Writes the `/etc/sec/rc_audit_events` file
- Writes the `/etc/sec/fs_objects` file

10.6.1.7 Audit Subsystem Startup Defaults

The system startup defaults for the audit subsystem are stored in AUDITMASK_FLAG in the `/etc/rc.config.common` file. This field is used to pass command-line arguments to `auditmask` at system startup. AUDITMASK_FLAG sets the audit style flags and sets `auditmask` to read the events selected for auditing from the `/etc/sec/rc_audit_events` file. The `/etc/sec/fs_objects` file contains a list of the files that have property lists that were modified during the audit subsystem configuration to allow the file to be monitored with object selection and deselection. Refer to the `auditmask`(8) reference page for more information on object selection and deselection.

`/etc/rc.config.common` AUDITMASK_FLAG

Can be modified with `rcmgr`, `sysman auditconfig` or with the Audit Manager through the CDE Dashboard.

`/etc/sec/rc_audit_events`

Can be modified using a text editor, `sysman auditconfig`, or with the Audit Manager through the CDE Dashboard.

/etc/sec/fs_objects

Modification of /etc/sec/fs_objects requires care. This file contains the only easily available list of files that have the object selection and deselection flags set in the file's property list. Any files that the system administrator needs to set the object selection and deselection flag on should be included in this file. The /etc/sec/fs_objects file can be modified by using system auditconfig.

The procedure for manually modifying this file is as follows:

Caution

While this procedure is being performed, your system may be vulnerable.

1. Execute the following commands to reset the object selection and deselection flags in the property list:

```
# auditmask -X /etc/sec/fs_objects:0
# auditmask -Y /etc/sec/fs_objects:0
```

2. Modify the /etc/sec/fs_objects file using a text editor.
3. If obj_sel is in AUDITMASK_FLAG, use:

```
# auditmask -X /etc/sec/fs_objects:1
```

If obj_desel is in AUDITMASK_FLAG, use:

```
# auditmask -Y /etc/sec/fs_objects:1
```

10.6.2 After the Data Has Been Collected

After the audit log data has been collected, it is important to have a structured archival procedure for the audit log files. Many of the applications that use the audit log data need to locate and analyze data that can be several months old.

10.6.2.1 Audit Log Trim Procedures

The sysman auditconfig utility provides a method for periodically deleting the binary audit logs using a root cron job. This feature is disabled by default. If enabled, the system administrator should provide for periodic audit log file backup before each cron job run.

The "Audit log lifespan in months" dialog box reflects both the monthly cycle on which the deletion will execute and the criteria for the deletion. The deletion always occurs on the first of the month; the hour is adjustable, but

the minute is hardwired to zero. A binary audit log is deleted when all the audit events it contains are older than the deletion execution date minus the “Audit log life span in months”.

For example, to create a `cron` job that runs on the second, fourth, and sixth months on the first day of the month at 3:00 a.m., enter the following:

```
"Audit log lifespan in months" = "every second month"
"Hour of deletion(0-23)" = "3"
```

Binary log files that contain entries that are all at least two months old are deleted at `cron` execution.

10.7 Stopping Audit

The audit daemon can be terminated using either of the following command lines:

```
# rcmgr -c delete AIDITMASK_FLAG
# rcmgr -c delete AIDITD_FLAG
```

or

```
# auditmask [-cluster] -n
# auditd [-cluster] -dk
```

10.8 Auditing Across a Network

If you have computers linked in a TCP/IP network, you can run the audit daemon on multiple systems and feed the information logged to a single system (the audit hub) for storage and analysis, as follows:

1. On the host that is to be the central collecting point for audit information (the audit hub), create the file `/etc/sec/auditd_clients`. Each line in this file must have the name of a remote host that will be feeding audit data to the local audit daemon.
2. On the audit hub, enter the following command to enable the audit hub to receive audit data from audit daemons on remote hosts specified in the `/etc/sec/auditd_clients` file:

   # **/usr/sbin/auditd –s**
3. On each remote host, direct the audit data to the system that is the audit hub with the following command:

   # **/usr/sbin/auditd –l *audit_hub_name*:**

   If communication is broken with the audit hub and it can no longer receive data, the local daemon stores the audit data locally, as specified with the `–o` and `–r` flags to `auditd`.
4. On the audit hub, you can set options for remote audit daemons as follows:

auditd [-p *ID_of_daemon_serving_remote_host options_for_remote_daemon*]

For example, to set the audit log location to `/var/audit/NYC_Sys1` for the remote host served by the audit daemon with ID 6:

```
# /usr/sbin/auditd –p 6 –l /var/audit/NYC_Sys1
```

The IDs of audit daemons serving remote hosts are integers. To learn the IDs, use the command:

```
# /usr/sbin/auditd –w
```

When feeding audit data from remote hosts to an audit hub, direct the audit data from each remote host into its own, dedicated audit log file on the hub system. This is necessary to prevent corruption of audit data, and is done by default.

When you use the audit tool to retrieve data from these logs of audit data from remote systems, the first and last audit log entries may be fragments rather than complete entries. This can happen if the communications channel is not cleanly terminated or if the `auditd` remotely receiving the data is forced to switch log files. This is because remote audit information is fed in a continuous stream to the audit hub, rather than as discrete audit entries.

The audit tool notifies you when it encounters a fragmented entry. This does not affect the retrieval of other records from the audit log.

10.9 Contents of Audit Records

All audit records, whether originating from system calls, trusted events, site-defined events, or the `audgen` command include, at a minimum, the following elements. The only exception is that if a login is not completed, `audit_id:` does not appear.

```
audit_id:                ruid/euid:
pid:                     ppid:              cttydev:
event:
result:
ip address:
timestamp:
```

`audit_id:` The Audit ID (AUID). The AUID is associated with a user at login and should remain unchanged throughout the login session. Processes started by the user will have the AUID associated with them. The AUID is not affected by use of the `su` command to change user IDs.

AUID and LUID (login UID) are synonyms.

Note that a malicious process that gains root access can use `setluid()` to change its audit ID. To monitor such a privileged process, enable monitoring of the security system calls. A `setluid()` call is identified in the audit record for the `security()` by the first argument being `0x3`. Both the original and new audit IDs are available from the audit record. The original audit ID is saved as the result code (return value) and the new audit ID is the argument.

`ruid/euid:` The real user ID (RUID) and the effective user ID (EUID).

`pid:` The process ID.

`ppid:` The parent process ID (PPID). Useful for tracing back through a list of processes to the originating event and its associated RUID and AUID.

`cttydev:` The device on which the event occurred. The record reports the major and minor numbers.

`event:` The name of the event (typically, either the name of a system call or the name of a trusted event).

`result:` or `error:` If the event succeeds, the result. Often the result is 0. But some system calls return a different value. For example `write()` returns the number of bytes written.

In the case of an error, the audit record for a system call returns the error message and number. For example,

```
error:     Not owner (1)
```

Audit records for trusted events can have additional error messages.

`ip address:` The IP address of the system on which event occurred.

`timestamp:` The date and time of the event.

10.9.1 Additional Entries in Audit Records

Most entries in an audit record for a system call are arguments to that system call. The following list presents many of the labels for entries that can go into an audit record, and brief explanations of what those labels can mean. But the labels are context sensitive. That is, their meaning can depend on the type of audit record in which they appear.

For example, in the audit record for `mmap()`, `flag:` indicates an attribute of the mapped region. But in an audit record for `audcntl()`, `flag:` is a number passed with a HABITAT request.

Commands entered on the command line by users appear as arguments to `char param:` in audit reports. For example, if a user copies the file `august_report` to a file named `sept_report`, the audit record includes the following:

```
event:  execve
char param:  /usr/bin/cp
char param:  cp august_report sept_report
```

In the context of a given audit record, interpreting the entries is a straightforward matter. If questions do arise, the reference page for the system call being audited will help clarify the report.

The following is a list of the audit record fields and the associated explanations of the fields:

`address:` Memory address, typically an argument to `mmap()`.

`char param:` A character string. The string can be the argument to an event or some other information relevant to the event. For example:

```
event:       open
char param:  /etc/zoneinfo/localtime
```

`cntl flag:` A control flag. For example, one of the flags to an `audcntl()` request.

`descriptor:` A file descriptor. If state-dependent information is available, the actual file name and the descriptor.

`devname:` A device name.

`directory:` The name of the current directory.

`flag:`	A `flag` argument to a system call. For example, in the context of `mmap()`, it specifies the attributes of the mapped region. In the context of `audcntl()`, it is the number of a system call, and it is passed with one of the HABITAT requests.
`flags:`	Arguments passed to system calls as flags. For example, in `open()`, the value passed as the `oflag` argument.
`gid:`	The group ID.
`home dir:`	The home directory.
`hostname:`	A host name.
`inode id:`	An inode number. Along with `inode dev`, part of the descriptor information recorded for audit records involving activities with files.
`inode dev:`	The major and minor inode device numbers.
`int param:`	An integer value. For example, in `setpgid()`, the `process_id` argument.
`len:`	An argument to `mmap()`. The length of a region of memory.
`login name:`	The login user name.
`long param:`	A value of type `long`.
`mask:`	A mask argument. For example, in `audcntl()`, the value passed with a SET_SYS_AMASK request.
`object mode:`	The protection mode of an object. For example, in `open()`, the mode of the file being opened.
`operation:`	A request to `audcntl()`, such as SET_SITEMASK.
`pgrp:`	A process group ID. For example, the `process_group_id` argument to `setpgrp()`.

`procname:`	The process name associated with a PID.
`prot:`	An argument to `mmap()`. The protections on a region of memory.
`req mode:`	The request mode. For example, in an `open()` `O_CREAT`, the protection mode for the new file.
`request:`	The security action requested in a call to `security()`.
`shell:`	The user's shell program. Typically this record element appears in `login` event records.
`username;`	The user name associated with the event. If the `audit_tool -w` option or the Audit Manager Translate UID/GIDs to Local Names selection was used to generate the audit report, then the user name might appear in parentheses. Parentheses indicate that the audit reduction tool had to use the `getpw()` routine to look up the user name in `/etc/passwd`. This happens in cases where the user name was not associated with the RUID at login time (for example, logins were not included among the audited events). See Section 10.5.3 for a description of dependencies among audit events.

10.9.2 Example Audit Record

The following is an example of an audit record:

```
audit_id:    1621       ruid/euid: 0/0   username: jdoe
pid:         5742       ppid: 1          cttydev: (39,0)
event:       login
login name:  jdoe
home dir:    /usr/users/jdoe
shell:       /bin/csh
devname:     tty02
char param:  Login succeeded
char param:  ZK33C5
directory:   /usr/users/jdoe
result:      0
ip address:  16.153.127.240 (alpha1)
timestamp:   Wed Jul 28 19:17:52.63 1996 EDT
```

10.9.3 Abbreviated Audit Records

The `audit_tool -B` command generates an audit report with an abbreviated record format.

An example of an abbreviated report:

```
AUID:RUID:EUID    PID    RES/(ERR)   EVENT
--------------    ---    ---------   -------------------------------------
-1:0:0            2056   0x0         execve (/usr/sbin/rlogind rlogind )
-1:0:0            2057   0x0         execve (/usr/bin/login login -p -h
                                             alpha1.sales.dec.com guest)
1234:0:0          2057   0x0         login  (guest)
1234:1234:1234    2057   0x0         execve (/bin/sh -sh)
1234:1234:1234    2058   0x0         execve (/usr/bin/stty stty dec)
```

Column headings for abbreviated reports:

`AUID:RUID:EUID`	The audit ID, real UID, and effective UID associated with the event.
`PID`	The process ID number.
`RES/(ERR)`	`RES` is the result number. Refer to the reference page for the specific system call for information about the result number. `(ERR)` is the error code, if an error occurred. For a list of error codes and their meanings, see the reference pages for `errno`(2).
`EVENT`	The event and arguments appear in the last column.

The following information does not appear in the abbreviated report:

- User name (If you want the username instead of the AUID:RUID:EUID, use the `audit_tool -wB` command.)
- PPID
- Device ID
- Current directory
- Inode information
- Symbolic name referenced by descriptors
- IP address
- Time stamp

See the description of the `-O` option in the `audit_tool(8)` reference page to generate a customized brief report.

10.10 More About Generating Audit Reports

Audit reduction lets you process and filter data stored in the audit log and display the audit information in a format you can read.

From the CDE Dashboard, select the following:

Sysman Applications→Daily Administration→Audit Manager→Reports→Generate Reports

When generating reports with the Audit Manager graphical interface, you create a selection file that specifies such things as the events, times of day, AUIDs and other attributes of the audit records that you want included in the audit report.

From the command line, enter the following commands:

```
# audit_tool [-options] audit file name
```

For audit logs generated on an ULTRIX system, enter:

```
# audit_tool.ultrix
```

10.10.1 Filtering Out Specific Audit Records

Just as object selection and object deselection help you manage the size of audit logs by means of preselection, the audit reduction deselection feature can help you manage the size of audit reports. It supports the use of a deselection file to filter out audit records that you do not want to see.

A deselection file consists of one or more lines. Each line specifies a deselection rule in the following format:

[*hostname audit_ID RUID event pathname flag*]

An asterisk (*) in a field is a wildcard, which always gives a match. A string ending with a plus sign (+) matches any string that starts with the designated string. The *flag* specifies read (r) or write (w) mode for open events.

For example, to filter out all `open` operations for read access on objects whose pathname starts with `/usr/lib/`, specify the following line in the file:

```
* * * open /usr/lib/+ r
```

The lines that you specify in the deselection file take precedence over other selection options. You can create multiple deselection files, but you can specify only one deselection file each time you perform audit reduction.

Deselection files can be used with the CDE Audit Manager and with the following command:

```
# audit_tool –d deselection_file
```

10.10.2 Targeting Active Processes

This section shows how the `auditmask` and `audit_tool` commands can be used to get real-time audit data about a running process.

You can audit a process in real time by using the `–p` option to `auditmask`. Example 10–1 shows how you might investigate a process started by a user logged in as `guest`.

Example 10–1: Sample Active Auditing Session

```
# ps -uguest -o user,pid,uid,comm          1
USER       PID   UID COMMAND
guest    23561  1123 csh
guest    23563  1123 ed

# auditmask -p 23563 open exec -c or       2
# auditmask -p 23563                       3
!  Audited system calls:
execv                      succeed  fail
exec_with_loader           succeed  fail
open                       succeed  fail
execve                     succeed  fail

!  Audited trusted events:

!  Audcntl flag: or


# auditd -d 5s -w                          4
Audit data and msgs:
  -l) audit data destination   = /var/audit/auditlog.hostname.001
  -c) audit console messages           = /var/audit/auditd_cons
  -d) audit data dump frequency        = 5s

Network:
  -s) network audit server status (toggle) = off
  -t) connection timeout value (sec)       = 4

Overflow control:
  -f) % free space before overflow condition = 10
  -o) action to take on overflow    = overwrite current auditlog

# audit_tool /var/audit/auditlog.hostname.001 -Bfw    5
USERNAME    PID    RES/(ERR)   EVENT
```

Example 10–1: Sample Active Auditing Session (cont.)

```
--------    ---    ---------   -----
jdoe         23563  0x4         open ( /etc/motd 0x0 )
jdoe         23563  0x4         open ( /etc/passwd 0x0 )
jdoe         23563  0x4         open ( /etc/ftpusers 0x0 )
jdoe         23563  0x4         open ( /etc/hosts 0x0 )
jdoe         23583  0x0         execve ( /usr/bin/sh sh -c ps )
jdoe         23583  0x5         open ( /usr/shlib/libc.so 0x0 )
jdoe       * 23592  0x0         execve ( /sbin/ps ps gax ) 6
jdoe         23599  0x0         execve ( /usr/bin/sh sh -c w )
jdoe         23599  0x5         open ( /usr/shlib/libc.so 0x0 )
jdoe       * 24253  0x0         execve ( /usr/ucb/w w )
jdoe         23563  0x4         open ( savethis 0x602 0640 )

Ctrl/C          7
--interrupt:  exit (y/[n])?  y
#
```

1 Find out what process user `guest` is running and also get the process ID and audit ID.

2 For PID 23563, set the auditmask to `open` and `exec`, and perform an `OR` operation with the system mask. Note that `exec` is an alias for `execv`, `exec_with_loader`, and `execve`.

3 Get the auditmask for the 23563 process.

4 Dump to the audit log every 5 seconds and also show the `auditd` configuration.

5 Display a continuous (`-f`) abbreviated (`-B`) audit report. Resolve AUIDs to corresponding usernames (`-w`).

Note that the name of the audit log was gotten from the results of the `auditd –w` command.

6 The asterisk (*) indicates an operation involving `setuid`.

7 Exit the `audit_tool` program with a Ctrl/C (auditing continues).

See the `auditmask`(8) reference page for more information.

10.11 Audit Data Recovery

In the event your system encounters a panic situation, the `crashdc` utility extracts any audit data left in the system at the time of the panic. The audit data is placed in the crash directory, in the `audit-data.`*n* file

The *n* is the crash number. If no audit data was present, the file is not created. The `audit-data.`*n* file can be processed with `audit_tool`.

It is possible for some audit records to appear in both the `auditlog` file and `audit-data.`*n*. It is also possible that the first audit record in `audit-data.`*n* may not be complete. The `audit_tool` utility marks this as a corrupted record. In this case, the audit record has already been written to the regular `auditlog`.

10.12 Implementation Notes

The following is information about the Tru64 UNIX auditing that you should be aware of:

- Some records show "NOTE: uid changed." This typically occurs in SETUID events, but may be seen anywhere when one thread changes the UID for all threads in a process (task).
- Audit records contain inode information and the file type of the object of the operation. So, for example, if a `chmod` command is specified for a symbolic link, the actual object referenced by the link is described.
- By design, some system calls can fail and not generate an audit record for the failure if the failure is not security-relevant. See Table 10–5 for a list of the calls.
- Only `TIOCSTI` operations are audited for the `ioctl` system call.
- Only `F_DUPFD`, `F_SETTIMES`, and `F_CNVT` are audited for the `fcntl` system call.

If the process audit control flag is set to USR, all `ioctl` and `fcntl` system calls are audited.

All security-relevant system calls can generate audit data, but when there is no security relevance, some system calls do not generate audit data. The conditions under which a particular system call does not generate an audit data are described in Table 10–5.

Table 10–5: System Calls Not Always Audited

System Call	Cause for No Audit Record
`close`	The system call failed because it was passed an invalid file descriptor.
`dup2`	The system call failed because it was passed an invalid file descriptor.
`execv, execve,`	`namei` lookups failed
`exec_with_loader`	A thread failed to terminate, or a handler callout aborted.

Table 10–5: System Calls Not Always Audited (cont.)

System Call	Cause for No Audit Record
`fcntl(*)`	The system call failed because it was passed an invalid file descriptor.
`ioctl(*)`	The system call failed because it was passed an invalid file descriptor.
`priocntlset()`	An invalid process was specified (ESRCH), or the call did not modify another process.
`reboot()`	Successful reboots are not audited. The `reboot()` call does not return from a successful reboot.
`security()`	No audit record for the `getluid` option. This option has no security relevance, and if it were audited, many audit records of no use would result.
`swapctl()`	Only `SC_ADD` forms of the call are audited. Other forms have no security relevance.
`uadmin()`	Only a failed A_REBOOT or A_SHUTDOWN is audited. In other cases, the system is rebooted and the system call does not return.

The system calls marked with an asterisk (*) typically generate audit data only for security-relevant options. When executing processes from `auditmask` with the `–e` or `–E` flag, however, all options generate audit data.

10.13 Responding to Audit Reports

Whenever you suspect an effort is being made to violate security, you should consider increasing the frequency of auditing. Additionally, you might want to tailor the list of events being audited to gather more specific information about the attempted violations. For example, if the attacks are against the file system, you might want to log all failed and successful file `open`s and `close`s, `link`s, `unlink`s, `chdir`s, `chmod`s, `chown`s, and other file-related activities.

When the audit trail implicates a specific authorized user in attempts to violate security, you can take the following steps:

- Talk with the user, reminding him or her of the importance of maintaining security and the need for all users to contribute to that effort.
- Restrict the user's access to the system by placing the user in a group of one.
- In extreme cases, deny the user system access by removing the user's account. This can be on a temporary or permanent basis.

- Audit the offending user's activities for indications that the user's behavior has changed. When you extract audit information, pay close attention to activities associated with the user's audit ID, UID, RUID, and user name.

  User-specific audits can be done from the screen: Audit Manager → Reports → Generate Reports, or with the `audit_tool` options `–a` *AUID*, `-u` *UID*, and `–U` *username*.

If the audit trail indicates attempts to violate security but points to no specific user, it is possible that you are faced with intrusion by an outsider. Your responses must then be directed to the system and the larger user community. In this case you can take the following steps:

- Have users change their passwords and inform them about the selection of safe passwords.
- Hold meetings with users to discuss the importance of system security.
- Increase physical security to make sure that only authorized users can gain physical access to the system.
- Perform backups of the file system more frequently, to minimize the damage if a break-in should occur and data on the system is lost or altered.
- If attacks seem to be coming in over a network, increase the auditing of network-related activities.

10.14 Using Audit to Trace System Calls

The audit mechanism can be used to troubleshoot system problems by collecting system call trace data.

Some differences exist between audit system call tracing and conventional system call tracing packages such as `truss` and `trace`. One difference is that audit system call tracing provides only the security-relevant arguments for each system call. See Section 10.14.3 to learn how to modify the kernel to get more data for a system call. Conventional trace packages attempt to capture all system call arguments.

Another difference is that audit system call tracing provides information unavailable from conventional tracing packages. Such information includes the following:

- Inode ID
- Thread ID
- File mode
- File descriptor to pathname translation

Also, audit works without requiring control of the target process.

10.14.1 Tracing a Process

To use the audit subsystem to trace a process, do the following:

1. Ensure that the audit subsystem is configured and running.
2. Initially, set the auditmask to audit no events:

   ```
   auditmask -n
   ```

3. Use the `auditmask -c` option to set the process `audcntl` flag as appropriate.
4. If tracing a currently running process, use the `auditmask –c usr` option to trace all options for these system calls.

The following examples demonstrate the use of the `auditmask` utility. These examples modify the process auditmask; unless specified, the process `audcntl` flag remains at its default setting of `OR`.

In the following example, audit records are created for everything done by the newly executed *command* program with its associated arguments:

```
# auditmask –E command argument
```

In the following example, audit records are created for failed `open` system calls and successful `ipc` events (defined in `/etc/sec/event_aliases`) for the newly executed `command` program:

```
# auditmask open:0:1 ipc:1:0 –e command arguments
```

In the following example, for PID 999, audit all (`–f`) events except `gettimeofday`:

```
# auditmask –p 999 –f gettimeofday:0:0
```

In the following example, get the set of events being audited for PID 999:

```
# auditmask –p 999
```

In the following example, set the `audcntl` flag of PID 999 to `usr`:

```
# auditmask –p 999 –c usr
```

In the following example, for all processes owned by the user with AUID 1123, audit all `ipc` events (the AUID is the same as the user's initial RUID):

```
# auditmask –a 1123 ipc
```

The `auditmask -h` command displays help for the command. See also the `auditmask`(8) reference page.

10.14.2 Reading the Trace Data

Use the following procedure to read the trace data collected by the audit mechanism:

1. Use `auditd` to flush any buffered audit data as follows:

   ```
   # auditd –dq
   ```

   The –q option gets the name of the data file

2. Examine the data file with the `audit_tool` utility as follows:

   ```
   # audit_tool `auditd –dq` –B
   ```

See the `audit_tool`(8) reference page for further information.

10.14.3 Modifying the Kernel to Get More Data for a System Call

The audit subsystem normally collects the following data:

- System call name
- Result
- Error
- Timestamp
- ID information
- Various arguments passed to the system call

Only the arguments that are of interest from a security perspective are recorded. If additional arguments are required, you can use `dbx` to change which arguments get recorded for any system call. For example, `flock` is system call #131, and takes as arguments a file descriptor and an option. To audit these arguments, enter the following `dbx` commands:

```
(dbx) a sysent[131].aud_param[0]='c'
99
(dbx) a sysent[131].aud_param[1]='a'
97
```

The first entry in the `aud_parm` array corresponds to the first system call argument, the second entry corresponds to the second system call argument, and so on. The `c` encoding indicates a file descriptor is recorded. The `a` encoding indicates an integer argument is recorded. The set of encodings is described in the `<sys/audit.h>` file.

10.15 Traditional UNIX Logging Tools

For security-relevant auditing, use the audit subsystem. Traditional UNIX operating system logging tools are available and do provide some auditing capabilities for the following categories of events:

- Local login and logouts
- File Transfer Protocol (FTP) logins
- External logins and logouts for TCP/IP (`rlogin` and `telnet`)
- External logins and logouts for DECnet (`dlogin` and `set host`)
- Failed logins
- Failed attempts to become superuser (the `su` command)
- Reboots and crashes
- `rsh` and `rcp` file transfer requests
- DECnet file transfer requests

Auditing for each of these categories involves a data file, that stores the pertinent information, and a method for viewing the stored data. In some cases this method is a specific command, such as `last` or `lastcomm`. In other cases the contents of the file are viewed directly, for example, with the `more` command.

The accounting data is stored in a number of different files. Table 10–6 lists those files in the `/var/adm` directory. The presence of specific log files on your system depends on which logging and accounting features you have enabled.

Table 10–6: Traditional UNIX Log Files in /var/adm

File Name	Security-Relevant Information
`wtmp`	Records all logins, logouts, and shutdowns. Use the `last` command to view this log.
`syslog.dated/`*`date`*`/daemon.log`	Messages generated by system daemons.
`syslog.dated/`*`date`*`/kern.log`	Messages generated by the kernel (for example, for system crashes).
`syslog.dated/`*`date`*`/lpr.log`	Messages generated by the line printer spooling system.
`syslog.dated/`*`date`*`/mail.log`	Messages generated by the mail system.
`syslog.dated/`*`date`*`/user.log`	Messages generated by user processes.

Table 10–6: Traditional UNIX Log Files in /var/adm (cont.)

File Name	Security-Relevant Information
`syslog.dated/`*`date`*`/syslog.log`	Requests for DECnet file transfers.
`acct`	Raw system accounting data, including user commands.

Protect the contents of these files. The files and directories should be owned by the root account, and they should not be writeable by `group` or `other`.

For a discussion of traditional UNIX accounting software, see the *System Administration* manual.

11

Administering ACLs

This chapter describes the installation and administration of the ACLs on a Tru64 UNIX system.

The Tru64 UNIX ACLs are based on the POSIX P1003.6 Draft 13 standard with some Draft 15 extensions.

For a technical description of ACLs, see Chapter 5 and the `acl`(4) reference page.

11.1 ACL Subsystem Overview

The ACL subsystem is shipped as part of the base system, but base system components do not require ACLs for current operations and no files are shipped with ACLs on them.

If layered products need ACL support, then ACLs must be enabled. If this is not done, then access granted to an object may not be the correct access.

Because ACLs are stored on a file system's property list (extended attributes), Tru64 UNIX supports ACLs on file systems that provide property lists. The supported files systems are as follows:

- Network file system (NFS) where both client and server are Tru64 UNIX systems. The property list daemon (`proplistd`) must be enabled on the server and the `-o proplist` option to the `mount` command must be used on the client.
- UNIX file system (UFS)
- Advanced file system (AdvFS)

ACLs on DFS, CDFS, and Internet IPC sockets are not supported.

See the `proplist`(4) and `proplistd`(8) reference pages for more information about property lists.

11.2 Administration Tasks

The primary tasks of the administrator relative to ACLs are:

- Enabling and disabling ACLs on the system.

- Creating new file control database entries for new applications that use ACLs when they are added to the system. See Section 6.5.2.4 for more information about this database.
- As superuser, modifying ACLs on behalf of users who are not authorized to access the associated files.
- Assigning ACLs when an imported file contains an ACL that cannot be converted to one that is recognized on the system.
- Creating and maintaining an ACL inheritance strategy for all files on your system needing ACL protection.
- Some system commands, applications, and user programs may check file permissions, create files, or modify files using old methods that do not process ACLs and other extended attributes stored in property lists. It is the responsibility of the system administrator to educate the programmers and users on the system about the use of ACLs in general and the specific ACLs in use on the system. For information on the impact to system users, see Section 5.8. For more information on writing or modifying programs to work properly with ACLs, see the `acl`(4) reference page and Chapter 21.

To administer the ACLs on your Tru64 UNIX system, you need to be familiar with the commands documented on the following reference pages:

`dxsetacl`(1)	Graphical interface to display and change discretionary access control information.
`getacl`(1)	Displays discretionary access control information.
`setacl`(1)	Changes the access control list on a file or directory.
`acl`(4)	Provides information about the Tru64 UNIX ACL implementation
`secconfig`(8), `sysconfig`(8)	Used to enable and disable the ACL subsystem.
`nfssetup`(8), `proplistd`(8), `proplist`(8)	Enable and disable the property lists on NFS filesystems.

11.3 Installing ACLs

The optional ACLs are shipped as part of the base system and can be configured and used independently of the enhanced security subset or other security features. The enhanced security subsets (OSFC2SECxxx and OSFXC2SECxxx) do not need to be installed on your system.

Before you configure ACLs, you need to answer the following questions:

- Which objects on your system need to be protected with ACLs?
- What level of access are you going to permit on your ACL-protected objects?

You must set an ACL for each object that you want to protect. See the `dxsetacl`(1), `setacl`(1), and `getacl`(1) reference pages for instructions on setting and retrieving ACLs. Directories also have two default ACLs that can be set. These default ACLs define what ACLs are inherited by new files and subdirectories created under them. See Section 5.7 for a description of the ACL inheritance rules.

11.3.1 Enabling and Disabling ACLs

ACL processing is enabled and disabled dynamically using the `sysconfig` utility or the `secconfig` menu. To enable ACL processing dynamically using the `sysconfig` command, enter the following:

```
# sysconfig -r sec acl_mode=enable
```

To disable ACL processing dynamically using the `sysconfig` command, enter the following:

```
# sysconfig -r sec acl_mode=disable
```

To view the current ACL processing mode using the `sysconfig` command, enter the following:

```
# sysconfig -q sec
```

To have ACLs enabled automatically as part of system startup, create a `stanza` file containing the ACL mode enable entry, for example:

```
sec:
    acl_mode = enable
```

Then use `sysconfigdb` to add it to the `/etc/sysconfigtab` file:

```
# sysconfigdb -m -f acl_mode.stanza sec
```

On subsequent reboots, ACL processing is enabled automatically.

Note

ACLs can be set on files even if ACL processing is not enabled on the system. However, when ACL processing is not enabled on

the system, ACLs will not be used in access checks. Also, if ACL processing is not enabled on the system, Default ACLs are not used and ACL inheritance is not done.

11.3.2 Enabling ACLs on NFS

For an NFS client to make direct use of ACLs or extended attributes (property lists) over NFS, the `proplistd` daemon must be enabled on an NFS server. The `proplist` mount option must be used when mounting on the client. Access checks are enforced by the server in any case, although NFSv2 client caching could sometimes cause inappropriate read access to be granted. Correctly implemented NFSv3 clients make the necessary access checks.

Start the `proplistd` daemon by selecting the number of `proplist` daemons to run when you use the setup GUI or the `nfssetup` utility. You can also start the daemons manually with the `proplistd` command. For example:

```
# /usr/sbin/proplistd 4
```

On the client, the file system must be mounted with the `proplist` option by either of the following methods:

- Add `proplist` to the options field in the `/etc/fstab` file:

  ```
  sware1:/advfs /nfs_advfs nfs rw,proplist 0 0
  ```

- Alternatively, add the `proplist` option to the `mount` command as follows:

  ```
  # mount -o proplist sware1:/advfs /nfs_advfs
  ```

11.4 Recovery

The `fsck` and `fsdb` commands are used to recover property lists and ACLs, respectively, in the event of a system crash. If ACLs are enabled when `fsck` is run on a file system, `fsck` verifies all property lists on the file system unless instructed otherwise. If a property list is found that is not correct, `fsck` attempts to correct it. In most cases, restoring the property list also restores the ACL. The ACLs are validated by kernel read for access decisions.

The `fsdb` command examines the ACL in either the internal or external format. A privileged user can change the ACL using `fsdb`.

11.5 Standalone System Support

Because the standalone system (SAS) is strictly intended for installing a system and repairing the root file system, the ACL code is not enabled. The

fsck and fsdb commands have the ability to extract and manipulate ACLs from the property list obtained from the raw partition.

11.6 Archival Tool Interaction with ACLs

The interaction of ACLs with the archiving tools is described in the following sections.

11.6.1 pax and tar

Both pax and tar archive any ACLs and other extended attributes on archived files and directories by default when you create an archive. However, when you use the pax or tar utilites to extract files and directories from an archive, any ACLs on the archived files and directories are not extracted from the archive by default. If the destination directory has default ACLs defined, the files and directories extracted from the archive inherit the default ACLs as described in Section 5.7.

To restore the ACLs and property list information from the archive, use the -p option for tar and the -p p or -p e options to pax when extracting files and directories from the archive. The pax and tar utilities store the user and group information for ACLs as UIDs and GIDs. This means that if you use the tar -p, pax -p p or pax -p e commands to restore an archive on a system that does not share user and group information with the source system you may be granting unintended access to files.

There currently are no formal industry standards for ACLs and extended attributes (property lists). Thus, the extensions to pax and tar to support property lists and ACLs are specific to Tru64 UNIX. Other vendor's pax and tar implementations should simply ignore the Tru64 UNIX specific extensions. However, to ensure interoperability with other vendor's systems, when archiving for multivendor distribution, use the -V option to prevent ACLs and any other extended attributes from being archived.

11.6.2 dump and restore

The archive tools dump, rdump, restore, rrestore, vdump, rvdump, vrestore, and rvrestore always save and restore all extended attributes (property lists) including ACLs. Attempting to extract files to a directory that has a default ACL or a default access ACL may cause unintended ACLs to be created on the extracted files. If ACLs are enabled on the system, make sure to check all ACLs after the extraction is complete.

11.7 ACL Size Limitations

On AdvFS file systems there is a hard limit of 1560 bytes for a property list entry. Because ACLs are stored in property list entries, this equates to

62 ACL entries in addition to the three required ACL entries. The error EINVAL is returned if you exceed this limit.

To facilitate interoperation of the UFS and AdvFS ACLs, a configurable limit is imposed on UFS ACLs. The default value of the UFS ACL limit is 1548 bytes (the AdvFS limit includes the header), equivalent to the 62+3 required entry limit on AdvFS.

The UFS configurable limit on ACLs is in the `sec` subsystem and has been given the attribute name `ufs-sec-proplist-max-entry`. The attribute can be dynamically configured using the `sysconfig` utility or by using `sysconfigdb` to set the attribute in the file `sysconfigtab`. A configurable property list element size for UFS has also been added to the `sec` subsystem and has been given the attribute name `ufs-proplist-max-entry`.

The value of `ufs-proplist-max-entry` must be larger than `ufs-sec-proplist-max-entry` by at least enough space to hold a property list element header. The `sysconfig` utility automatically adjusts `ufs-proplist-max-entry` to achieve this. The default value of `ufs-proplist-max-entry` is 8192 bytes. See the `cfgmgf`(8), `seconfig`(8), `seconfigdb`(8), and `sysconfigtab`(4) reference pages for more information.

12

Ensuring Authentication Database Integrity

The authck program, which checks the internal consistency of the files that make up the authentication database, is used to verify the integrity of your system. This function cannot be performed with the GUIs.

This chapter describes the authck program, suggests reasons for running it, and explains what to do if it finds discrepancies.

12.1 Composition of the Authentication Database

The enhanced security authentication database, consists of the following subsidiary databases:

- Protected password database (/tcb/files/auth.db and /var/tcb/files/auth.db)
- System defaults database (/etc/auth/system/default)
- Terminal control database (/etc/auth/system/ttys.db)
- File control database (/etc/auth/system/files)
- Device assignment database (/etc/auth/system/devassign)

For detailed information about the format and contents of the databases, see the default(4), devassign(4), files(4), prpasswd(4), and ttys(4) reference pages.

The system management GUI interface is the preferred method for modifying these databases. But, for disaster recovery or for times when the GUI interface is not available, the edauth program can be used to modify the databases. In single user mode, the /usr and /var file systems must be mounted before the edauth program is used.

12.2 Running the authck Program

The authck program checks the overall structure and the internal consistency of the authentication database. The authck program checks for the correctness of entries within each database and also checks related fields in other databases. For example, it checks the protected password database entry for a user against the /etc/passwd file.

You can specify the following arguments on the `authck` command line:

`–p`	Checks the protected password database and the `/etc/passwd` file to ensure that they are complete and that they agree with each other. It also checks the protected password database for reasonable values.
`–t`	Checks the fields in the terminal control database for reasonable values.
`–f`	Checks the file control database for syntax and value specification errors. Without this flag, entries with unknown authorizations, user names, and so on, are ignored. Typically these errors are typographical, such as “rooot” instead of “root,” and the program attempts to guess the right value.
`–v`	Verbose mode.
`–a`	Performs the functions of `–f`, `–p`, `–t`, and `–v`. Provides program activity status during operation.

The `authck` program produces a report listing any discrepancies between the databases. Compare the output of the program with the actual database entries and rectify any differences immediately. Problems typically occur because someone has manually updated one of the databases without making the corresponding change to the related databases.

12.3 Adding Applications to the File Control Database

When you add applications to the system by a means other than the `setld` program, you should also add file control database entries for the application’s control and database files and programs. It is best to consult with the application supplier to get a file and program list, and suggested protection attributes for all files.

If you add the application’s files to the file control database, you gain the benefit of periodic integrity checking of that application’s resources.

See the `fverify`(8) reference page for more information on checking file integrity.

12.4 Recovery of /etc/passwd Information

If the /etc/passwd file is lost, but the enhanced profiles are still available, then a command sequence like the following can be used to recover some of the missing data:

```
# bcheckrc
# /tcb/bin/convuser -dn | /usr/bin/xargs /tcb/bin/edauth -g | \
    sed '/:u_id#/!d;s/.*:u_name=//;s/:u_id#/:*:/;s/:u_.*$/:/' \
        > psw.missing
```

This creates a psw.missing file containing entries like the following:

```
root:*:0:
jdoe:*:0:
```

Primary group, finger, home directory, and login shell information is not recorded in the enhanced profile. The data for those fields must be recovered by other means.

13

Security Integration Architecture

This chapter describes the Tru64 UNIX security framework called the Security Integration Architecture (SIA). The chapter discusses the following topics:

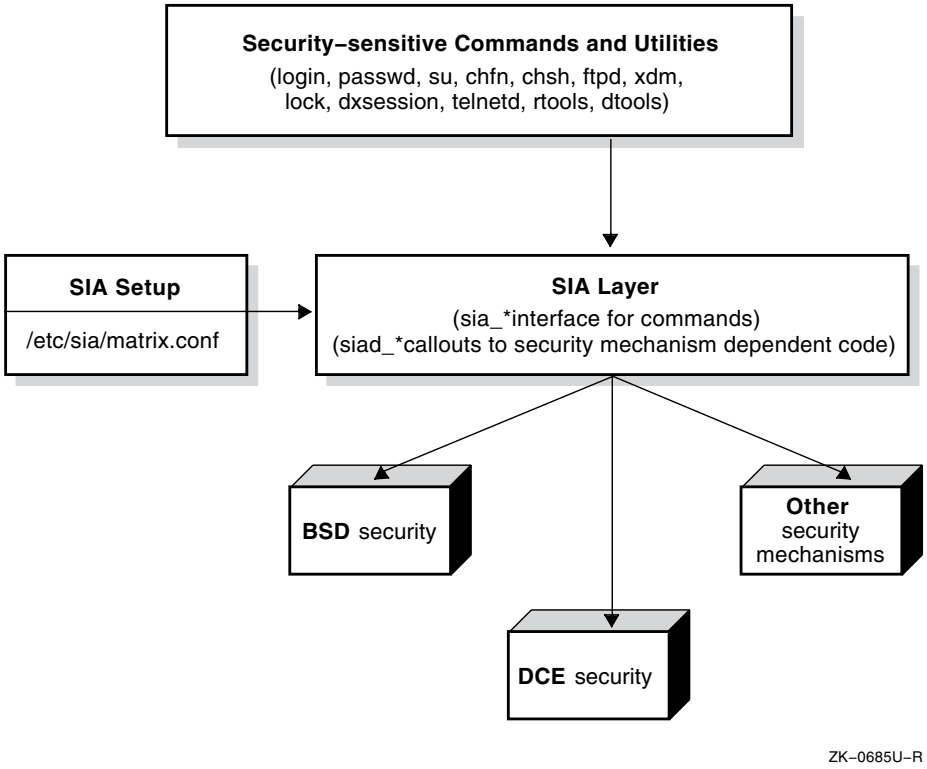
- Overview of the SIA
- Supported security configurations
- SIA's `matrix.conf` file
- Installation and deletion of layered security products

13.1 SIA Overview

All security authentication mechanisms that run on the Tru64 UNIX operating system run under the Security Integration Architecture (SIA) layer. The SIA allows you to layer various local and distributed security authentication mechanisms onto Tru64 UNIX with no modification to the security-sensitive Tru64 UNIX commands, such as `login`, `su`, and `passwd`. The SIA isolates the security-sensitive commands from the specific security mechanisms, thus eliminating the need to modify them for each new security mechanism.

Any time a security mechanism is installed or deleted, the SIA is involved. You do not need to be concerned about the SIA layer if you do not install security products. Each time that a security-sensitive command is invoked, the SIA layer serves as an interface to code that depends upon security mechanisms.

Figure 13–1: Security Integration Architecture



13.2 Supported Security Configurations

The Tru64 UNIX operating system provides standard Berkeley security (BASE), which is limited to `/etc/passwd` local security with NIS extensions, and the optional enhanced security (ENHANCED), which includes enhanced password features. Audit capability and ACLs can be enabled independently of enhanced security.

13.3 matrix.conf Files

The security configuration file that selects the appropriate installed security mechanism is the `matrix.conf` file. The system is provided with a default base (BSD) security `matrix.conf` file (`/etc/sia/matrix.conf`). The `siacfg` utility is used to automatically update `matrix.conf` for additional security mecahanisms. Example 13–1 shows the default BSD `matrix.conf` (`/etc/sia/matrix.conf`) file:

Example 13–1: Default /etc/sia/matrix.conf File

```
siad_init=(BSD,libc.so)
siad_chk_invoker=(OSFC2,libsecurity.so)
siad_ses_init=(OSFC2,libsecurity.so)
siad_ses_authent=(OSFC2,libsecurity.so)
siad_ses_estab=(OSFC2,libsecurity.so)
siad_ses_launch=(OSFC2,libsecurity.so)
siad_ses_suauthent=(OSFC2,libsecurity.so)
siad_ses_reauthent=(OSFC2,libsecurity.so)
siad_chg_finger=(OSFC2,libsecurity.so)
siad_chg_password=(OSFC2,libsecurity.so)
siad_chg_shell=(OSFC2,libsecurity.so)
siad_getpwent=(BSD,libc.so)
siad_getpwuid=(BSD,libc.so)
siad_getpwnam=(BSD,libc.so)
siad_setpwent=(BSD,libc.so)
siad_endpwent=(BSD,libc.so)
siad_getgrent=(BSD,libc.so)
siad_getgrgid=(BSD,libc.so)
siad_getgrnam=(BSD,libc.so)
siad_setgrent=(BSD,libc.so)
siad_endgrent=(BSD,libc.so)
siad_ses_release=(OSFC2,libsecurity.so)
siad_chk_user=(OSFC2,libsecurity.so)
```

See the `matrix.conf`(4) and `siacfg`(8) reference pages for more information.

13.4 Installing a Layered Security Product

Detailed instructions for installing layered security products are provided by the layered product. In general, you install a layered security product as follows:

1. Install the layered security product as described in the product's installation procedure.
2. Change directory to `/etc/sia`.
3. Run the `siacfg` utility.
4. Reboot your system.

13.5 Installing Multiple Layered Security Products

The Tru64 UNIX operating system supports the installation of multiple security products.

Detailed instructions for installing multiple layered security products is provided by the layered products. In general, you install multiple layered security products as follows:

1. Bring the system down to single-user mode using the `/usr/sbin/shutdown now` command.
2. Install the first layered security product as described in the product's installation procedure.
3. Install the subsequent layered security product, as described in the product's installation procedure.
4. Change directory to `/etc/sia`.
5. Run the `siacfg` utility for each layered security product.
6. Reboot your system.

13.6 Removing Layered Security Products

To remove a layered security product from your system, perform the following steps:

1. Verify that the installed layered security product has not changed the BSD security mechanism or associated files. This information is usually described in the documentation that came with the product.

   Note

   If the BSD security mechanism cannot be restored (for example, the `/etc/passwd` file has been deleted), then the operating system must be reinstalled and reconfigured.

2. Bring the system down to single-user mode using the `/usr/sbin/shutdown now` command.
3. Remove the link to the layered security product's `matrix.conf` file using the `siacfg -r` command.
4. Reboot your system.

Example 13–2 shows how to delete a DCE layered security product and return to BASE security.

Example 13–2: Changing a Layered Security Product

```
# /usr/sbin/shutdown now
# /sbin/siacfg -r DCE
# /sbin/siacfg -l BSD libc.so
# /usr/sbin/reboot
```

13.7 SIA Logging

SIA will optionally record the success and failure of security-related commands in the `/var/adm/sialog` file. If the file exists, log entries are made by SIA. This procedure is recommended for debugging only.

14

Trusted System Troubleshooting

This chapter describes problems that can occur on your system and gives guidance on how to avoid or correct from them. It provides you with insight on what is involved in the system startup, so you can examine critical files and programs required for correct system operation. Once the system is in single-user mode, there is no substitute for careful backup procedures. This is the only precaution that will avert serious data loss in your system.

The problems discussed in the following sections will prevent the system from booting. Chapter 12 demonstrates authentication database verification.

14.1 Lock Files

The system security databases are critical to correct system operation. These databases use a lock file to synchronize rewrites to security-relevant databases. Before a process rewrites a database entry, it automatically creates the lock file. If the lock file already exists, the program assumes that another process is currently using the database and waits for the lock file to be removed. If the lock file persists and is not modified within a reasonable time period (currently 50 seconds), the program waiting for the lock file removes it and creates a new one, assuming that there has been a system crash or software error.

The system names lock files by appending a `:t` extension to the normal file name.

The system's startup scripts include lines that remove all lock files at system startup. The following files have associated lock files that can prevent correct operation of the system:

- `/dev/console`
- `/etc/auth/system/default`
- `/etc/auth/system/devassign`

14.2 Required Files and File Contents

The following files are required to run the system:

- `/tcb/files/auth.db`

- /etc/auth/system/ttys.db
- /etc/auth/system/default
- /etc/auth/system/devassign
- /etc/passwd
- /etc/group
- /sbin/rc[023]s
- /dev/console
- /dev/tty*
- /dev/pts/*
- /sbin/sulogin
- /sbin/sh
- /vmunix

14.2.1 The /tcb/files/auth.db Database

When the system begins operation, it consults the security databases for various parameters. If any of the databases are corrupt, the system will not boot successfully. If possible, the startup programs report that there is a problem in the databases and start a single-user shell at the system console to allow you to repair the system. In some cases, however, the system will not boot and you must repair the system from standalone procedures described in the manual *System Administration*.

The enhanced (protected) password database entry for root is held in the /tcb/files/auth.db database. If the entry for root is inconsistent, the system enters single-user mode, but assumes default characteristics for all security parameters of the shell it starts.

When the system is in single-user mode, you can create an enhanced (protected) password database entry for root by entering the following command:

```
# edauth root
```

The following example shows a typical enhanced (protected) password database entry for root:

```
root:u_name=root:u_id#0:\
    :u_pwd=encrypted_password:\

    :u_minchg#0:u_pickpw:u_nullpw:u_restrict@:\
    :u_maxtries#100:u_lock@:chkent:
```

For a complete explanation of all the fields, see prpasswd(4). The following fields are required for the system to be able to boot:

name	Must contain root.
u_name	Must also be root.
u_uid	Must have a value of 0.
u_pwd	The encrypted version of the password. At authentication, the system checks the entered password against the encrypted version of the password. You can leave this field blank if you are creating the database entry.
chkent	As with all databases, the entry must end with the single word *chkent*.

The other fields in this entry are informational or are used to guard against unwanted account locking. The system overrides all conditions that can cause the root account to lock when changing to single-user mode.

14.2.2 The /etc/auth/system/ttys.db File

The terminal control database must have a valid entry for the system console. The entry for the system console must begin with the word console followed by a colon. It must end with the single word chkent. The only required field is t_devname, which must be set to a value of console. For example:

```
console:t_devname=console:chkent:
```

14.2.3 The /etc/auth/system/default File

The system default database must have an initial field default and must end with chkent. There must not be a :t lock file associated with this database.

The following example is typical:

```
default:\
        :d_name=default:\
        :d_boot_authenticate@:\
        :d_audit_enable@:\
        :d_pw_expire_warning#3456000:\
        :u_pwd=*:\
        :u_minchg#0:u_maxlen#20:u_exp#15724800:u_life#31449600:\
```

```
:u_pickpw:u_genpwd:u_restrict@:u_nullpw@:\
:u_genchars:u_genletters:u_maxtries#5:u_lock@:\
:t_logdelay#1:t_maxtries#5:t_lock@:t_login_timeout#60:\
:chkent:
```

14.2.4 The /etc/auth/system/devassign File

If the entry for the console is inconsistent, no application can be started. The field must start with the word `console` and end with the word `chkent`. The *v_type* field must be set to `terminal`.

The following example is typical:

```
console:v_devs=/dev/console:v_type=terminal:\
       :chkent:
```

14.2.5 The /etc/passwd File

The `/etc/passwd` file is the password database. This file must be present and its format must be correct. No encrypted passwords are updated in this file.

14.2.6 The /etc/group File

The `/etc/group` file is the group database. This file must be present and its format must be correct.

14.2.7 The /sbin/rc[023] Files

The `/sbin/rc[023]` files are used by `init` to change between run levels. Save copies of these files after installation.

14.2.8 The /dev/console File

The `/dev/console` file designates the character device associated with the system console. This file must be present for the system to boot.

14.2.9 The /dev/pts/* and /dev/tty* Files

The `/dev/pts/*` and `/dev/tty*` files are pseudoterminal devices used for interprocess communication.

14.2.10 The /sbin/sulogin File

The `/sbin/sulogin` executable file allows restricting access in single-user mode to those users with the root password.

14.2.11 The /sbin/sh File

The /sbin/sh executable file must be present for the system to start a shell to transition to single-user mode.

14.2.12 The /vmunix File

The /vmunix file is the executable image of the operating system. The boot loading software loads the operating system into memory and transfers control to it at boot time.

14.3 Problems Logging In or Changing Passwords

If users experience problems logging in to the system or changing their passwords, examine the file attributes for the files in the security subset using the fverify command. For example, to verify the file attributes for the files in the OSFC2SEC510 subset, enter the following commands:

```
# cd /
# /usr/lbin/fverify < /usr/.smdb./OSFC2SEC510.inv
```

The file attributes of the local user profile files are examined using the ls –l and authck –pf commands.

If a user complains of login troubles involving the inability to update the protected profile or to obtain a lock and you are running centralized account management, see Section 9.5.

The utilities such as dxaccounts and usermod share a lock file called /etc/.AM_is_running. If the file is present, the utilities warn you.

Part 3

Programmer's Guide to Security

15

Introduction for Programmers

This chapter describes the implication of running trusted applications on a trusted Tru64 UNIX system. Libraries, header files, the standard trusted system directories and the trusted computing base (TCB) are discussed. This chapter and the ones that follow use partial and complete C programs to illustrate basic ideas. Although some of these can be used without modification, they are not a collection of routines from which you can assemble trusted programs.

15.1 Libraries and Header Files

Your system documentation contains reference pages for all security system calls (section 2) and routines (section 3).

The `libsecurity.so`, `libaud.a`, `libaud.so`, `libpacl.a`, and the `libpacl.so` libraries hold the enhanced security interface binaries. Use the `-lsecurity` compilation option to link these into your program, for example:

```
$ cc ... -lsecurity -ldb -lm -laud ...
```

Your programs need to include several header files that hold definitions (constants, macros, structures, library interfaces, and so forth) necessary to use the Tru64 UNIX security interfaces. Following traditional UNIX practice, all Tru64 UNIX system call and library reference pages denote the header files that you need to use their routines. You are likely to use the following individual header files, in the order listed:

`<sys/secdefines.h>`	Defines compilation constants that determine the security configuration of your system. You always need to include this file first.
`<sys/security.h>`	Holds general definitions. You almost always need to include this file.
`<sys/acl.h>`	For access control lists. You need this if you manipulate access control lists.
`<prot.h>`	Defines the authentication databases and Tru64 UNIX protected subsystems. You

	need these if your program accesses any of the authentication databases.
`<sys/audit.h>`	Defines the audit subsystem constants for security audit interfaces. You need this if you generate or process audit records.
`<protcmd.h>`	Provides a few miscellaneous definitions for trusted commands that are delivered with Tru64 UNIX. You seldom need these.
`<sia.h>`	SIA constants, structures, and macro definitions
`<siad.h>`	SIA constants, structures, and macro definitions internally used by the interfaces and security mechanisms

15.2 Standard Trusted System Directories

Tru64 UNIX defines several directories to hold its security information. You can review the reference pages for a description of these files and directories, primarily the section 4 reference pages.

You may need to create new files and directories in the standard trusted system directories. Generally, you should create new directories for the files you place in these trees. Do not simply insert new files in existing directories unless that directory was explicitly created for such files. Table 15–1 lists the directories you might use:

Table 15–1: Standard Trusted System Directories

Directory	Contents
`/tcb/bin, /usr/tcb/bin`	Contains directly executed trusted commands and daemons.
`/tcb/lib`	Contains programs that are run by other trusted programs but are never invoked from the command line.
`/tcb/files`	Contains control files, databases, and scripts used by the trusted computing base (TCB). You can define a subdirectory of this directory for your protected subsystem, if necessary.
`/var/tcb`	Alternative to the `/tcb` directory.

15.3 Security-Relevant System Calls and Library Routines

The tables in the following sections list many of the Tru64 UNIX system calls and library routines that have security implications for programmers.

Note that some system calls and library routines not covered in these sections might also have implicit security concerns.

The misuse of a system call or library routine that does not seem to have any security concerns could threaten the security of a computer system. For example, all system calls bypass file access permissions when called by a privileged process. Ultimately, programmers are responsible for the security implications of their programs.

15.3.1 System Calls

Table 15–2 lists the system calls that have security relevance for programmers.

Table 15–2: Security-Relevant System Calls

Category	System Calls
File control	`creat`, `open`, `fcntl`, `read`, `mknod`[a], `write`
Process control	`fork`, `sigpause`, `execve`, `sigsetmask`, `setpgrp`[a], `sigvec`, `sigblock`
File attributes	`access`, `chroot`[a], `chmod`[a], `stat`, `chown`[a], `umask`
User and group ID	`getegid`, `getuid`, `getgid`, `setgroups`[a], `geteuid`, `setreuid`[a]
Auditing	`audcntl`[a], `audgen`[a]
General	`syscall`

[a] These system calls can be called only by a privileged process or they may behave differently when called by a nonprivileged process. See the associated reference pages for more information.

15.3.2 Library Routines

Library routines are system services that programs can call. Many library routines use system calls. Table 15–3 lists Tru64 UNIX library routines that have security implications.

Table 15–3: Security-Relevant Library Routines

Category	Library Routines
File control	`fopen, popen`

Table 15–3: Security-Relevant Library Routines (cont.)

Category	Library Routines
Password handling	`getpass, putpwent, getpwnam, setpwent, getpwent, endpwent, getpwuid, passlen, pw_mapping, randomword, time_lock`
Process control	`signal`

15.4 Defining the Trusted Computing Base

You must protect the trusted computing base (TCB) from unintended modification. To do this, you first define which of your programs and data files are a part of the TCB. The following list describes the components of the TCB:

- *Trusted Programs:* Any program that could subvert a security rule must be considered a trusted program. This includes programs that make direct security decisions, and those that do not, but could subvert security if they contained errors or malicious code. Consider a program trusted if the program file has its user ID set to root (SUID).
- *Indirect Programs:* A program is trusted if another trusted program invokes it or otherwise interacts with it and depends upon its actions for security decisions. A program is also trusted if it modifies a data file or other object upon which another trusted program depends.
- *Program Files:* Executable files that contain a trusted program are considered a part of the TCB.
- *Object Code and Libraries:* All object (binary) code modules and their files, whether statically or dynamically linked, that are included in a trusted program are part of the TCB. This includes the standard C library routines and interfaces, which are frequently used by trusted programs.
- *Data Files:* The TCB includes any file that contains data used by a trusted program to make a security decision, for example, the `ttys` database.
- *Shell Scripts:* A shell script is a data file that a shell program interprets, performing the shell commands in the file. A shell script is considered part of the TCB if it performs a function on behalf of a trusted program or if it is needed for correct operation of the system. You can determine if a shell script is security relevant if removing or replacing the script would cause the system to perform improperly (for example, removing some of the `rc` startup scripts) or provide an opportunity for a security breach (installing a different `cron` startup file). Shell script files should be protected as carefully as object code program files. Note that a shell script must be readable to be executed.

- *Antecedent Directories:* Consider all parent directories of TCB files a part of the TCB and protect them accordingly. If malicious users can remove and redefine links in these directories, then they can create new, phony files that might cause a trusted program to make an incorrect security decision.

15.5 Protecting TCB Files

Each of the following mechanisms presents a way to protect the files and directories of the TCB:

- *Discretionary Access Control (DAC):* Discretionary access control (the owner, group, mode bits, and ACLs) is the most important protection for TCB files. It must prevent untrusted users and groups from modifying these files, although they might be allowed to read the files. It is common to create pseudousers and pseudogroups for this purpose.

  Existing programs may copy only the mode bits when replicating a file and therefore accidentally delete the ACL. This removes the protection offered by the ACL. Compaq recommends that you use restrictive traditional permissions, such as `other::---` and `group::---`, and then grant access to individual users with user entries. Using this approach, if an ACL is lost, unintended access is not allowed. See Chapter 21 for information on programming with ACLs.
- *Read-Only File Systems:* You can place all files that only need to be read on a separate file system and mount that file system as read only. This ensures that no program, no matter how privileged, can alter those files (at least short of remounting the file system). You can, of course, remount the file system as read/write if you need to alter the files. This is somewhat drastic but offers good protection against corruption of security data. You can also physically set a read only locking tab on many kinds of removable media.
- *Sticky Bit:* Tru64 UNIX includes the sticky bit on directories. The sticky bit restricts the removal of directory entries (links) to those owned by the requesting user or the owner of the directory. Without this protection, programs only need write access to the directory. Use the sticky bit where appropriate; for example, when a program needs to store files owned by different users in a single directory.

16

Trusted Programming Techniques

This chapter presents specific techniques for designing trusted programs.

16.1 Writing SUID and SGID Programs

SUID (set user ID) and SGID (set group ID) programs change the effective UID or GID of a process to the UID or GID of the program. They are a solution to the problem of providing controlled access to system-level files and directories, because they give a process the access rights of the files' owner.

The potential for security abuse is higher for programs in which the user ID is set to `root` or the group ID is set to any group that provides write access to system-level files. Do not write a program that sets the user ID to `root` unless there is no other way to accomplish the task.

The `chown` system call automatically removes any SUID or SGID bits on a file, unless the RUID of the executing process is set to zero. This prevents the accidental creation of SUID or SGID programs owned by the `root` account. For more information, see `chown`(2).

The following list provides suggestions for creating more secure SUID and SGID programs:

- Verify all user-provided pathnames with the `access` system call.
- Trap all relevant signals to prevent core dumps.
- Test for all error conditions, such as system call return values and buffer overflow.

When possible, create SGID programs rather than SUID programs. One reason is that file access is generally more restrictive for a group than for a user. If your SGID program is compromised, the restrictive file access reduces the range of actions available to the attacker.

Another reason is that it is easier to access files owned by the user executing the SGID program. When a user executes an SUID program, the original effective UID is no longer available for use for file access. However, when a user executes an SGID program, the user's primary GID is still available as part of the group access list. Therefore, the SGID process still has group access to the files that the user could access.

The stack of all SUID programs is not executable by default. User applications that rely on the stack being executable will fail. If absolutely necessary, you can change the default setting. This allows the stack of SUID programs to be executable. To change from default of zero (not executable) to executable, use the following command:

```
# sysconfig -r proc executable_stack=1
```

To ensure that the change persists across reboots, use the `sysconfigdb` command to add the entry to the `/etc/sysconfigtab` file.

16.2 Handling Errors

Most system calls and library routines return an integer return code, which indicates the success or failure of the call. Always check the return code to make sure that a routine succeeded. If the call fails, test the global variable `errno` to find out why it failed.

The `errno` variable is set when an error occurs in a system call. You can use this value to obtain a more detailed description of the error condition. You can use this information to determine how your program will respond or to produce helpful diagnostic messages. This error code corresponds to an error name in `<errno.h>`. For more information, see `errno`(2).

The following `errno` values indicate a possible security breach:

EPERM	Indicates an attempt by someone other than the owner to modify a file in a way reserved to the file owner or superuser. It can also mean that a user attempted to do something that is reserved for a superuser.
EACCES	Indicates an attempt to access a file for which the user does not have permission.
EROFS	Indicates an attempt to access a file on a mounted file system when that permission has been revoked.

If your program makes a privileged system call but the resulting executable program does not have superuser privilege, it will fail when it tries to execute the privileged system call. If the security administrator has set up the audit system to log failed attempts to execute privileged system calls, the failure will be audited.

If your program detects a possible security breach, do not have it display a diagnostic message that could help an attacker defeat the program. For instance, do not display a message that indicates the program is about to

exit because the attacker's real user ID (UID) did not match a UID in an access file, or even worse, provide the name of the access file. Restrict this information by using the `audgen()` routine for SUID root programs and using `syslog` for other programs. In addition, you could program a small delay before issuing a message to prevent programmed attempts to penetrate your program by systematically trying various inputs.

16.3 Protecting Permanent and Temporary Files

If your program uses any permanent files (for example, a database), make sure these files have restrictive permissions and that your program provides controlled access. These precautions also apply to shared memory segments, semaphores, and interprocess communication mechanisms; set restrictive permissions on all of these objects.

Programs sometimes create temporary files to store data while the program is running. Follow these precautions when you use temporary files:

- Be sure your program deletes temporary files before it exits.
- Avoid storing sensitive information in temporary files, unless the information has been encrypted.
- Give only the owner of the temporary file read and write permission. Set the file creation mask to 077 by using the `umask()` system call at the beginning of the program.
- Create temporary files in private directories that are writable only by the owner or in `/tmp`. The `/tmp` directory has the sticky bit set (mode 1777), so that files in it can be deleted only by the file owner, the owner of the directory, or the superuser.

A common practice is to create a temporary file, then unlink the file while it is still open. This limits access to any processes that had the file open before the unlink; when the processes exit, the inode is released.

Note that this use of `unlink` on an NFS-mounted file system takes a slightly different action. The client kernel renames the file and the unlink is sent to NFS only when the process exits. You cannot guarantee that the file will be inaccessible to someone else, but you can be reasonably sure that the file will be inaccessible when the process exits. In any case, always explicitly ensure that no temporary files remain after the process exits.

16.4 Specifying a Secure Search Path

If you use the `popen`, `system`, or `exec*p` routines, which execute `/bin/sh` or `/sbin/sh`, be careful when specifying a pathname or defining the shell `PATH` variable. The `PATH` variable is a security-sensitive variable because

it specifies the search path for executing commands and scripts on your system. For more information, see environ(7), popen(3), and system(3).

The following list describes how to create a secure search path:

- Specify absolute pathnames for the PATH variable.
- Do not include public or temporary directories, other users' directories, or the current working directory in your search path. Including these directories increases the possibility of inadvertently executing the wrong program or of being trapped by a malicious program.
- Be sure that system directories appear before user directories in the list. This prevents you from mistakenly executing a program that might have the same name as a system program.
- Analyze your path-list syntax, especially your use of nulls, decimal points, and colons. A null entry or decimal point entry in a path list specifies the current working directory and a colon is used to separate entries in the path list. For this reason, the first entry following an equal sign should never begin with a colon.
- If a path list ends with a colon, certain shells and exec*p routines search the current working directory last. To avoid having various shells interpret this trailing colon in different ways, use the decimal point rather than a null entry to reference the current working directory.

You might want to use the execve system call rather than any of the exec*p routines because execve requires that you specify the pathname. For more information, see execve(2).

16.5 Responding to Signals

The Tru64 UNIX operating system generates signals in response to certain events. The event could be initiated by a user at a terminal (such as quit, interrupt, or stop), by a program error (such as a bus error), or by another program (such as kill).

By default, most signals terminate the receiving process; however, some signals only stop the receiving process. Many signals, such as SIGQUIT or SIGTRAP, write the core image to a file for debugging purposes. A core image file might contain sensitive information, such as passwords.

To protect sensitive information in core image files and protect programs from being interrupted by input from the keyboard, write programs that capture signals such as SIGQUIT, SIGTRAP, or SIGTSTP.

Use the signal routine to cause your process to change its response to a signal. This routine enables a process to ignore a signal or call a subroutine when the signal is delivered. (The SIGKILL and SIGSTOP signals cannot

be caught, ignored, or blocked. They are always passed to the receiving process.) For more information, see `signal`(3) and `sigvec`(2).

Also, be aware that child processes inherit the signal mask that the parent process sets before calling `fork`. The `execve` system call resets all caught signals to the default action; ignored signals remain ignored. Therefore, be sure that processes handle signals appropriately before you call `fork` or `execve`. For more information, see the `fork`(2) and `execve`(2) reference pages.

16.6 Using Open File Descriptors with Child Processes

A child process can inherit all the open file descriptors of its parent process and therefore can have the same type of access to files. This relationship creates a security concern.

For example, suppose you write a set user ID (SUID) program that does the following:

- Allows users to write data to a sensitive, privileged file
- Creates a child process that runs in a nonprivileged state

Because the parent SUID process opens a file for writing, the child (or any user running the child process) can write to that sensitive file.

To protect sensitive, privileged files from users of a child process, close all file descriptors that are not needed by the child process before the child is created. An efficient way to close file descriptors before creating a child process is to use the `fcntl` system call. You can use this call to set the `close-on-exec` flag on the file after you open it. File descriptors that have this flag set are automatically closed when the process starts a new program with the `exec` system call.

For more information, see the `fcntl`(2) reference page.

16.7 Security Concerns in the X Environment

The following sections discuss several ways to increase security in the X programming environment:

- Restrict access control
- Protect keyboard input
- Block keyboard and mouse events
- Protect device-related events

16.7.1 Protect Keyboard Input

Users logged into hosts listed in the access control list can call the `XGrabKeyboard` function to take control of the keyboard. When a client has called this function, the X server directs all keyboard events only to that client. Using this call, an attacker could grab the input stream from a window and direct it to another window. The attacker could return simulated keystrokes to the window to fool the user running the window. Thus, the user might not realize that anything was wrong.

The ability of an attacker to capture a user's keystrokes threatens the confidentiality of the data stored on the workstation.

The X Windows System provides a secure keyboard mode that directs everything a user types at the workstation keyboard to a single, secure window. Users can set this mode by selecting the Secure Keyboard item from the Commands menu in an X window.

Include a secure keyboard mode in programs that deal with sensitive data. This precaution is especially important if your program prompts a user for a password.

Some guidelines for implementing secure keyboard mode follow:

- Use the `XGrabKeyboard` call to the `Xlib` library.
- Use a visual cue to let the user know that secure keyboard mode has been set; for example, reverse video on the screen.
- Use the `XUngrabKeyboard` function to release the keyboard grab when the user reduces the window to an icon. Releasing the keyboard frees the user to direct keystrokes to another window.

16.7.2 Block Keyboard and Mouse Events

Hosts listed in the access control list can send events to any window if they know its ID. The `XSendEvent` call enables the calling application to send keyboard or mouse events to the specified window. An attacker could use this call to send potentially destructive data to a window. For example, this data could execute the `rm -rf *` command or use a text editor to change the contents of a sensitive file. If the terminal was idle, a user might not notice these commands being executed.

The ability of an attacker to send potentially destructive data to a workstation window threatens the integrity of the data stored on the workstation.

The X Windows System blocks keyboard and mouse events sent from another client if the `allowSendEvents` resource is set to `False` in the `.Xdefaults` file.

You can write programs that block events sent from other clients. The `XSendEvent` call sends an event to the specified window and sets the `send_event` flag in the event structure to `True`. Test this flag for each keyboard and mouse event that your program accepts. If the flag is set to `False`, the event was initiated by the keyboard and is safe to accept.

16.7.3 Protect Device-Related Events

Device-related events, such as keyboard and mouse events, propagate upward from the source window to ancestor windows until one of the following conditions is met:

- An X client selects the event for a window by setting its event mask
- An X client rejects the event by including that event in the `do-not-propagate` mask

You can use the `XReparentWindow` function to change the parent of a window. This call changes a window's parent to another window on the same screen. All you need to know to change a window's parent is the window ID. With the window ID of the child, you can discover the window ID of its parent.

The misuse of the `XReparentWindow` call can threaten security in a windowing system. The new parent window can select any event that the child window does not select.

Take these precautions to protect against this type of abuse:

- Have the child window select the device events that it needs. This precaution prevents the new parent from intercepting events that propagated upward from the child. Parent windows that centralize event handling for child windows are at greater security risk. An attacker can change the parent and intercept the events intended for the children. Therefore, it is safer for each child window to handle its own device events. Events that the child explicitly selects never propagate.
- Have the child window specify that device events will not propagate further in the window hierarchy by setting the `do-not-propagate` mask. This precaution prevents any device event from propagating to the parent window, regardless of whether the child requested the event.
- Have the child window ask to be notified when its parent window is changed by setting the `StructureNotify` or `SubstructureNotify` bit in the child window's event mask. For information on setting these event masks, see the *X Window System: The Complete Reference to Xlib, X Protocol, ICCCM, XLFD*.

16.8 Protecting Shell Scripts

When you write a shell script that handles sensitive data, set and export the `PATH` variable before writing the body of the script. Do not make shell scripts SUID or SGID.

17

Authentication Database

The authentication database is a set of databases that store all Tru64 UNIX security information when enhanced security is enabled. The following databases comprise the authentication database:

- Device assignment
- File control
- System default
- Protected password
- Terminal control

This chapter introduces each database and discusses its logical organization.

The trusted programs (that is, any program that could subvert a security rule) you create specifically for systems with enhanced security enabled need to use the information in these databases. Except for a few specialized cases, system administrators maintain these databases using the Tru64 UNIX administrative interfaces. Therefore your programs usually only read them. This chapter describes the databases only to the extent that they are used by your programs. See the system management chapters of this book for information on managing these databases. The `authcap`(4) reference page contains general information on the file format.

17.1 Accessing the Databases

Tru64 UNIX includes a set of library routines to access each database. The following reference pages describe the form and use of these databases; you should read them with this chapter.

Subject	Database	Reference Page
Device Assignment	`devassign`	`getesdvent`(3)
File Control	`file`	`getesfient`(3)
System Default	`default`	`getesdfent`(3)
Protected Password	`auth.db/prpasswd (NIS)`	`getespwent`(3)
Terminal Control	`ttys.db`	`getestcent`(3)

The library routines defined on these reference pages hide the actual file format of the databases. Trusted programs do not need to know the format; they simply use these library routines.

17.2 Database Components

Each database consists of a set of named entries. Programs primarily use the name of the entry to request a specific entry from a database, although a program can also sequentially search through the entries in a database.

Each entry contains a set of fields. Each field has an identifier, used to access the field and a value. Each database has an allowed set of fields in one of its entries. Individual fields are optional and can be omitted from an entry. There are several types of fields including string, integer, and Boolean.

The general format for an entry is as follows:

```
entry_name:string_field=value:integer_field#value:\
                        :boolean_field_true:boolean_field_false@:chkent:
```

In general, library routines read or write an entry as a whole. A C structure holds all possible fields for a given entry of the database. This structure is always accompanied by a flags structure which holds a mask designating which fields are to be read or written.

Your programs should take appropriate action when a field is undefined. In many cases, the undefined fields should be fetched from the system defaults database, as described in Section 17.2.1. Structures for each database include system default fields and flags for that database. Thus, it is easy to retrieve the system default values associated with a particular field because the system default values are available from the same structure that stores values for the individual entry.

17.2.1 Database Form

In general, you will not have to deal with the physical format of the authentication databases. All databases have the same logical form and similar access libraries. For example, the terminal control database consists of an entry for each controlled terminal. The following `ttys` file sample physical format entry for `tty01` and the associated table illustrate the database file format.

```
tty01:t_devname=tty01:t_uid#44:t_logtime#772479074:\
 :t_login_timeout#20:t_failures#3:t_lock@:\
 :chkent:
```

Meaning	Field	Value	Description
Name	t_devname	tty01	Terminal 1
User of last login	t_uid	44	UID of 44
Time of last login	t_logtime	772479074	Fri Jun 24 13:31:13 EDT 1994
Login timeout	t_login_timeout#20	20	Login timeout in seconds
Attempts since last login	t_failures#3	3	Failed login attempts since last successful login
Account status	t_lock	@	Unlocked (false)
Check entry	:chkent:<EOL>	chkent	End of entry

The following C structure is used for fetching an entry from the `ttys` database (see the include file `<prot.h>`):

```
struct es_term  {
    struct estc_field  *ufld;     /* fields for this entry */
    struct estc_flag   *uflg;     /* flags for this entry */
    struct estc_field  *sfld;     /* system default fields */
    struct estc_flag   *sflg;     /* system default flags */
};
```

The `estc_field` holds the data for the fields of the entry, and `estc_flag` holds the flags that designate which fields in `estc_field` are present or are set. The following is the `estc_field` structure:

```
struct estc_field  {
    char    *fd_devname;      /* terminal name */
    uid_t   fd_uid;           /* uid of last successful login */
    time_t  fd_slogin;        /* time of last successful login*/
    ushort  fd_nlogins;       /* consecutive failed attempts */
    char    fd_lock;          /* terminal lock status */
    ushort  fd_login_timeout; /* login timeout value */
};

struct estc_flag  {
unsigned short
    fg_devname        :1,     /* name present? */
    fg_uid            :1,     /* uid present? */
    fg_slogin         :1,     /* time present? */
    fg_nlogins        :1,     /* failed attempts present? */
    fg_lock           :1,     /* lock status present? */
    fg_login_timeout  :1      /* login timeout present? */
```

```
};
```

The `getestcent(3)` reference page defines the library routines that you can use to access the terminal control database. The access routines return or set the fields for a specific entry `ufld` and `uflg` and for the system defaults (`sfld` and `sflg`). For each database whose fields have system defaults, the system defaults are returned in addition to the fields for that entry.

17.2.2 Reading and Writing a Database

Each database is owned by a user/group, to which your program must have discretionary access. Your program can be installed in two ways:

- SGID to the appropriate group as a standard program of the subsystem
- SUID 0 as a standard program of the subsystem

The library routines automatically enforce one database writer at a time. However, the database is locked only for the duration of the time the database is being rewritten. There is no way to lock an entry against access across a retrieval and write operation. You should complete your writes as quickly as possible.

17.2.2.1 Buffer Management

You must understand how the system allocates and returns buffers for database entries to properly code programs that retrieve, replace, and add database entries. All database routines are patterned after the `getpwent()` routines in that they return pointers to static storage that is reused on each call. You must save the buffer contents if you are going to retrieve another entry and need to refer again to the previous entry, or if you need to rewrite an existing entry or add a new entry. You cannot read a database entry, change one or more field and flag values, and submit the same buffer to the routine that modifies the database.

The logical form for some database entry fields is self-contained. Other fields contain pointers to variable length data.

The `devassign` database logical form contains some fields that are pointers to variable length data. The `getesdvent(3)` reference page describes the `copyesdvent()` routine that allocates a structure to hold a device assignment database entry and copies the contents of a buffer returned from `getesdvent()` or `getesdvnam()` into it.

You can save an entry for a self-contained database by simple structure assignment, as follows:

```
struct es_passwd *pr;       /* returned value */
struct es_passwd *pwcopy;   /* buffer for saved values */
```

```
/* Retrieve john's protected password database entry */

pr = getesnam("john");

/* store values of john's entry to a local buffer */

pwcopy = copyespwent(pr);
if (!pwcopy) abort();

/* Change the password minimum change time to two weeks */

pwcopy->uflg->fg_min = 1;
pwcopy->ufld->fd_min = 14 * 24 * 60 * 60;

/* Rewrite john's protected password database entry */

if (!putespwnam("john", pwcopy))
    errmsg("Could not write protected password entry\n");
free(pwcopy);
```

17.2.2.2 Reading an Entry by Name or ID

You can read database entries by specifying their name or, in some databases, some other identifying value. For example, you can fetch entries from the enhanced (protected) password database by the entry name (the user's name) or the user ID. The following code reads the entry associated with the name `tty44` from the terminal control database:

```
 .
 .
 .
struct es_term   *entry;
 .
 .
 .
if ((entry = getestcnam("tty44")) == NULL)
            errmsg ("Entry not found");
```

Note that `getestcnam()` allocates the data structure for the returned entry. Hence, `entry` is only a pointer to an `es_term` structure that is reused the next time any of the `prtc()` or `estc()` routines is called.

17.2.2.3 Reading Entries Sequentially

You can also read database entries sequentially as illustrated in the following code:

```
 .
 .
 .
```

```
struct es_term   *entry;
 .
 .
 .
setprtcent();                                /* rewind the database*/
while ((entry = getestcent()) != NULL){      /* read next entry   */
 .                                           /* process the entry */
 .
 .
}
endprtcent ();                               /* close */
```

Note that `getestcent()` also allocates the data structure for the entry. You can restart the search from the beginning using `setprtcent()`.

17.2.2.4 Using System Defaults

A system default is a field that is used when the corresponding field in an entry is not defined. The system default database contains defaults for the other databases. The following databases contain information for which there are system defaults:

- Protected password
- Terminal control
- Device assignment

Note that only certain fields in these databases are allowed to have defaults.

When your program reads a logical entry, the library routine returns both the fields for that entry (`ufld` and `uflg`) and for the system default (`sfld` and `sflg`). If the entry does not contain the field you need, use the system default. In some cases if the system default is also undefined, your program should generate audit data to report the error and execute a failure path. In other cases, you can safely define a default value.

For example, if you need to determine the timeout value for the terminal `tty14`, your code might look like this:

```
struct es_term  *entry;        /* the entry for the terminal */
ushort          time_out;      /* final timeout value */
  .
  .
  .

/*--- fetch the entry by name ---*/

if ((entry = getestcnam ("tty14")) == NULL)
      errmsg ("Entry not found");

/*--- if defined for the terminal, use it ---*/
```

```
if (entry->uflg->fg_login_timeout)
      time_out = entry->ufld->fd_login_timeout;

/*--- else if system default is defined, use it ---*/

else if (entry->sflg->fg_login_timeout)
      time_out = entry->sfld->fd_login_timeout;

/*--- otherwise, assume a value of 0 ---*/

else time_out = 0;
```

17.2.2.5 Writing an Entry

Your program should seldom have to modify a database, and even more rarely a system default. However, if this is necessary, place the new fields in `ufld` and set the corresponding flags in `uflg`, and then call the appropriate library routine. For example, to set a new timeout value for the terminal `tty14` to 20, your code might look like this:

```
struct es_term  *entry, *ecopy;
 .
 .
 .

/*--- fetch the entry by name ---*/

if ((entry = getestcnam("tty14")) == NULL)
      errmsg ("Entry not found");

/*--- change the desired field(s) ---*/

ecopy = copyestcent(entry); if (! ecopy) abort():
ecopy->ufld->fd_login_timeout = 20;  /* set timeout value */
ecopy->uflg->fg_login_timeout =  1;  /* set flag to show the
                                        field has been set */
/*--- update the database ---*/

if (!putestcnam("tty14", ecopy))
      errmsg ("Could not update database");
free(ecopy);
```

Note

You must call the appropriate `copyes*()` routine to save the data for later use.

The copyes*() routines return pointers to a malloc() storage area that the caller must clear.

You can only set system defaults using the putesdfnam() interface for the system default database. You cannot, for example, set the sfld and sflg fields in an es_term entry and then call putestcnam() to set system defaults.

17.3 Device Assignment Database (devassign)

The device assignment database contains device attributes for devices on the system. There are two kinds of devices included in the devassign database:

- Terminals
- X displays

The name of a device entry is used in the device-related commands. This name is independent of the names of the device files that represent the device.

System administrators maintain the device assignment database; your programs should not modify its contents.

The logical entries for this database have dynamic sizes (are not self-contained). For this reason, you must use the copyesdvent() routine to make a working copy of a structure that contains one of its entries. See the getesdvent(3) reference page for details.

The text file /etc/auth/system/devassign holds the entire device assignment database.

17.4 File Control Database

The file control database helps to assure that your security-sensitive files have the correct protection attributes (owner, mode bits, and so forth). It contains the absolute pathname and the correct attributes for each file (or directory). These attributes include any combination of the following:

- File type (regular, block special, character special, directory, fifo, socket)
- Owner
- Group
- Permission mode bits
- Access control list (if the system is configured for access control lists)

Your programs should not read from or write to the file control database other than to use its entries for newly created files through the

create_file_securely() interface. However, you should add all new security-sensitive files and directories to the database. Include all of the attributes that do not change. This ensures that these attributes are regularly checked and corrected.

You can use the create_file_securely() routine to create files with the attributes specified in the file control database. This routine can only be used to create a new file. You should create new versions of files in a different file. (The Tru64 UNIX convention is to append a :t to a pathname for the file's new contents.) Then rename the new file (using the rename() system call) to the existing file name.

The file control database is a text file: /etc/auth/system/files. See the files(4) reference page for a definition of the format of this file. The system administrator can use the edauth –df command to add or remove entries from this database. See the edauth(8) reference page for more information.

17.5 System Default Database

The system default database, /etc/auth/system/default, is a text file that contains fields that are to be used when the corresponding fields are left undefined in other databases. Specifically, this database contains default information for the enhanced (protected) password, device assignment, and terminal control databases. (Note that all fields in each of the authentication databases may be left undefined, but all fields do not have system default values.)

The system default database also contains fields for miscellaneous system parameters. Your programs should not need this miscellaneous information.

System administrators maintain this database and your programs should never have to modify it. The access routines for other databases also return the system default values. See Section 17.7, for an example of how to access and use the information in the system default database.

The entire system default database has only one entry, the default entry.

17.6 Enhanced (Protected) Password Database

The enhanced password database (/tcb/files/auth.db and /var/tcb/files/auth.db) are dbm files that hold a set of user authentication profiles. User authentication profiles can also be distributed between Tru64 UNIX systems using the NIS prpasswd map. Each authentication profile entry is named with a user name (a name that a user supplies during login). The authentication profile has many fields that govern the user's login session. Chapter 18 describes these fields in detail.

An authentication profile is associated with the account whose presence is indicated by a line in the traditional /etc/passwd file or NIS passwd map. The encrypted password has been moved from the /etc/passwd file to the authentication profile.

The system assigns the traditional meanings for the other fields in the /etc/passwd database. Each entry in /etc/passwd corresponds to exactly one authentication profile in the protected password database with the same user ID and name. (Both must be present for an account to be considered valid.) The /etc/passwd entry contains a dummy encrypted password field; the authentication profile holds the real one.

The traditional UNIX interfaces for querying the /etc/passwd file is getpwent(). The interfaces' functions are unchanged and always fetch their information from the /etc/passwd file or NIS map. Note however, that the encrypted password that is returned is a dummy value. (The routine is not modified to retrieve the encrypted password from the authentication profile.)

Your programs should not modify the enhanced (protected) password database. However, many trusted programs need to read the information from the user authentication profiles.

17.7 Terminal Control Database

The terminal control database, /etc/auth/system/ttys.db, is a dbm file that contains fields used primarily during login that apply to the login terminal, as opposed to the user who is logging in. This database consists of an entry for each terminal upon which users may log in including X terminals.

Each entry in the database has a name of the terminal that matches a name in the file used to specify login ports (/etc/inittab). The entries in the device assignment database correspond to each entry in the terminal control database. Most trusted programs (for example, login) do not provide their services if there is no corresponding entry in the device assignment database.

Each terminal control database entry contains the following fields:

- The name of the terminal.
- The user ID and time of the last unsuccessful login attempt. Because the user ID is stored in the database, an unsuccessful login attempt that specifies a user name that does not map to a user ID does not produce a valid user ID in this database. If the user name maps to a valid ID, that ID is placed in the appropriate field.
- The user ID and time of the last successful login.
- The number of unsuccessful login attempts since the last successful login.

- Whether the terminal is locked.
- The number of unsuccessful attempts that the system allows before locking the terminal.
- The enforced time delay after a failed login attempt (enforced by the login program).
- The number of seconds after which the login, once started, times out if there is no keyboard input. Upon timeout, the login program terminates the login attempt.

System administrators maintain the entries in the terminal control database, although the Tru64 UNIX login programs modify many fields. Your programs do not usually modify this database. Although it is unlikely, trusted programs may need to read this database.

The file `/etc/auth/system/ttys.db` holds the entire terminal control database.

18

Identification and Authentication

This chapter discusses the following topics:

- The audit ID and some guidelines for using it
- The support libraries
- Using daemons
- The user authentication profile in the enhanced (protected) password database for enhanced security
- Some brief cautions for handling passwords

18.1 The Audit ID

Tru64 UNIX preserves all traditional UNIX process user and group identities. Additionally, it provides the per-process audit ID (AUID), which is unique to Tru64 UNIX. The AUID is similar in principle to the real user ID, except that it remains unchanged even in cases where the real user ID changes.

The audit ID is associated with all audit records and establishes the user identity even in those cases where the real and effective user IDs have been changed from their values at login.

The audit ID can be set only once in a line of process descendants, regardless of any process privileges. The audit ID is set at login to the authenticated user (the same as the real and effective user IDs) and is inherited from parent to child when a process forks using the `fork()` system call.

Programs that are created from startup scripts or that are created as a result of *respawn* entries in the `inittab` file are created with an unset audit ID. Such programs are normally authentication programs (`getty/login` sequences, window managers, trusted path managers) that set the AUID based on the user that authenticates through that interface.

Programs started through startup scripts typically receive requests for service on behalf of users and spawn a process to service that request. Such programs typically set the audit ID in the child service process based on the requesting process's effective identity. If you are writing this type of program, you should use the SIA routines. The SIA routines properly set

up the user's environment in the child process regardless of the security mechanisms in use on the system (BASE, enhanced, DCE, and so forth).

The `getluid()` and `setluid()` system calls read and set the audit ID. See their reference pages for details.

18.2 Identity Support Libraries

The Tru64 UNIX operating system provides several library routines for managing user and group identities. For example, the `set_auth_parameters()` routine is required by some routines used by enhanced security. It stores the initial user and group IDs that can later be queried or tested by the other routines. If you are writing a program or routine that will be used with the enhanced security option, you must call `set_auth_parameters()` at the beginning of your program's `main()` routine.

Several of the enhanced security routines for querying the authentication database require the program to have previously called `set_auth_parameters()` before changing any of the user or group IDs, or the command arguments *argc* and *argv*.

See the `identity(3)` reference page for more information.

To keep your code portable between security mechanisms, use the SIA session routines.

18.3 Using Daemons

Whenever a daemon performs an operation at the request of a user program (the client), it acts in one of two ways:

- It can run under its own identities, authorizations, and privileges, making its own decisions about what actions the requesting program may or may not perform. In this case, it does not need to change any of its own user identities.
- It can have the underlying operating system enforce operations as if the daemon had the client's security attributes (user IDs, authorizations, and so forth).

In the latter case, the daemon needs to establish a set of security attributes. The preferred technique is to fork a process, set the identities and privileges using SIA, and then either perform the actions directly or execute a program to perform them.

18.4 Using the Enhanced (Protected) Password Database

Although the enhanced (protected) password database is intended mainly for Tru64 UNIX programs, your programs may need to use the fields

described in the following list. (These fields are also described in the getespwent(3) and prpasswd(4) reference pages, the prot.h include file, and the administrative part of this document.)

- User name (u_name) and ID (u_id) — These fields correspond to the user name and ID in /etc/passwd.
- Encrypted password (u_pwd) — This field is the real encrypted password.
- Retired status (u_retired) — This field indicates whether the authentication profile is valid. If not valid, login sessions are not allowed. Once retired, an account should never again be reused.
- Login session priority (u_priority) — The process priority assigned to programs of the user login session using setpriority().
- User audit mask (u_auditmask) and control flags (u_audcntl) — This mask and its control flags, with the system audit mask, designate the events audited during the login session. The login program assigns a mask to the user's login shell. Audit masks and the control flags are inherited across exec() and fork() calls. See Chapter 19 and the auditmask(8) reference page for more information.
- Password parameters — The following parameters describe the login password and its generation:
  - Maximum length in characters for passwords chosen by the user (u_maxchosen)
  - Password expiration interval (u_exp)
  - Minimum password lifetime (u_minchg)
  - Password lifetime (u_life)
  - Time and date of last successful password change (u_succhg)
  - Time and date of last unsuccessful password change attempt (u_unsucchg)
  - User who last changed the password (u_pwchanger)
  - Password generation parameters (u_genpwd)
  - User generated password generation parameters (u_pickpw)
- Login password requirements (u_nullpw) — This is sometimes called the "null password option" and controls attempts to set a null password. Most administrators do not allow this option.
- Times during which a user may login (u_tod) — This field is formated like the UUCP systems file. (The systems file describes when a remote system can be contacted for file transfer.) It determines the valid times for a user to log in.

- Time and date of last login (u_suclog) — Expressed as a canonical UNIX time (in seconds since 1970).
- Terminal used during last login (u_suctty) — The terminal name is a cross-reference to the device assignment and terminal control databases.
- Number of unsuccessful login attempts since last login (u_numunsuclog) — This value is used to compute whether the terminal is disabled due to too many unsuccessful attempts.
- Number of unsuccessful login attempts allowed before disabling (u_maxtries) — This value is the user-specific limit for the number of unsuccessful attempts allowed until the account is disabled.
- Lock status (u_lock) — Whether or not the administrator has locked the account. A locked profile cannot be used for login or other services. Only an explicit request from the system administrator should unlock an authentication profile, and only programs that handle such requests should reset the locked field. A common programming error is to assume that the lock indicates all lock conditions. This indicator shows only the status of the administrative lock. An account may appear to be locked due to being disabled by password lifetime expiration or exceeding the number of unsuccessful attempts allowed for the account.

Your program can assume that with enhanced security enabled, the user name and ID in the enhanced (protected) password database is maintained by the system to have a corresponding entry in the /etc/passwd file.

18.5 Example: Password Expiration Program

The program named myexpire in Example 18–1 is a program for use with enhanced security that prints the user's password expiration time as defined in the enhanced (protected) password database. This program is part of the authentication protected subsystem and runs in the set group ID (SGID) mode, setting the GID to auth.

Example 18–1: Password Expiration Program

```
#include <sys/types.h>
#include <stdio.h>
#include <sys/security.h>
#include <prot.h>

main (argc, argv)
int        argc;
char      *argv[];
{
   struct es_passwd  *acct;
   time_t expire_time;
   time_t expire_date;
```

Example 18–1: Password Expiration Program (cont.)

```
   /*--- Standard initialization ---*/

   set_auth_parameters(argc, argv);
   initprivs();

   /*--- fetch account information using audit ID ---*/

   if ((acct = getespwuid(getluid())) == NULL)
      errmsg("Internal error");

  /*-- test if personal or system default applies and print --*/

   if (acct->uflg->fg_expire)
      expire_time = acct->ufld->fd_expire;
   else if (acct->sflg->fg_expire)
      expire_time = acct->sfld->fd_expire;
   else {
      audit_db_error(acct);     /* audit (externally defined) */
      errmsg("No user-specific or system default \
                                             expiration time.");
   }

   if (!acct->ufld->fg_schange) {
      audit_db_error(acct);      /* audit (externally defined) */
      errmsg("Account does not have successful change time");
   }

   expire_date = acct->ufld->fd_schange + expire_time;

   if (acct->uflg->fg_psw_chg_reqd && \
                acct->ufld->fd_psw_chg_reqd) \
                         expire_date = time((time_t *) NULL);

   audit_action(acct->ufld->fd_name, expire_date);
   exit(0);
}
```

Note

The enhanced (protected) password database files are accessible only to processes in the `auth` group. Programs that need to read the enhanced password database files must set the group ID to `auth`. (See the `setgid`(2) reference page.) To write this

information you must set the UID to 0 or to a user ID and have a group ID of `auth`.

18.6 Password Handling

Tru64 UNIX has been designed so that trusted programs can authenticate their users without specifically asking for passwords. Tru64 UNIX explicitly uses the audit ID for this purpose. Additional password handling is usually not necessary and difficult to handle securely. Appendix C provides an example of a program for password checking.

19

Audit Record Generation

This chapter provides information on the mechanics of writing and reading audit records. The following topics are covered:

- Audit events.
- Audit records and tokens.
- Audit flag and masks.
- Disabling auditing for the current process. (See Chapter 10 for information on managing the audit subsystem.)
- Modifying auditing for the current process.
- Generating application-specific audit records.
- Creating site-specific audit events.
- Creating your own audit logs.
- Parsing audit logs. (This section provides the low-level detail needed to develop additional utilities for audit data analysis.)

19.1 Introduction

Trusted programs can use the `audgen()` system call, the `audgenl()` library routine, or the `audgen` command to generate audit records; `audgenl()` is a front-end to `audgen()`. For arguments, the program supplies an audit event followed by audit data consisting of audit tokens and values.

The following code fragment shows how a program that checks boot authentication can call `audgenl()` to audit authentication failure:

```
if(audgenl(AUTH_EVENT, [1]
           AUD_T_LOGIN, pr->ufld.fd_name, [2]
           AUD_T_UID, pr->ufld.fd_uid,
           AUD_T_CHARP, "boot authentication failed"),0)== -1)
perror("audgenl");
```

Notes:

[1] AUTH_EVENT is the record event name.

[2] AUD_T_LOGIN, AUD_T_UID, and AUD_T_CHARP are tokens, each with a corresponding value.

These identifiers are defined in <sys/audit.h>. See Section 19.2 and Section 19.3 for descriptions of events and tokens.

19.2 Audit Events

Each audit record has an audit event associated with it. The system automatically adds the event when generating system call audit records. Self-auditing application programs pass the event as an argument to audgen() or audgenl() when generating audit records. There are two types of audit events available to application programs:

- Trusted events, which are defined in <sys/audit.h> with values between MIN_TRUSTED_EVENT and (MIN_TRUSTED_EVENT + N_TRUSTED_EVENTS -1). For example, the LOGIN event.
- Site-defined events, which are defined in /etc/sec/audit_events with values between MIN_SITE_EVENT and 1048576. The default range for site-defined events is 64. For information on defining site events, see Section 19.8.

19.3 Audit Records and Tokens

The audit subsystem has no fixed record type. Instead, an audit record is a series of tuples (data objects containing two or more components). Each tuple consists of an audit token and its corresponding value; depending on the token type, the tuple might contain a length field.

The following sections describe the two types of tokens: public tokens and private tokens. Application programs use public tokens.

19.3.1 Public Tokens

Public tokens are available to application programs that generate audit records using audgen() and audgenl(). Public tokens are defined in <sys/audit.h> and begin with AUD_T_; for example, AUD_T_CHARP.

There are three basic types of public tokens:

pointer	Used to represent data strings or structures as pointers. AUD_T_CHARP (character string) and AUD_T_HOMEDIR (home directory) are two pointer-type tokens.
iovec	Used to represent data as iovec-formatted data. AUD_T_OPAQUE, and AUD_T_INTARRAY are two iovec-type tokens. (Look for the iovec comments in <sys/audit.h>. The iovec structure is defined in <sys/uio.h>. For information about iovec, see the readv(2) and writev(2) reference pages.)
fixed length	Used to represent data as a 32- or 64-bit quantity. (AUD_T_RESULT and AUD_TP_LONG are 64–bit; others are 32-bit.) Most tokens use fixed-length data. AUD_T_AUID (audit ID), AUD_T_UID (user ID), and AUD_T_PID (process ID) are examples of fixed-length tokens.

The following example generates an audit record using iovec-formatted data:

```
#define AUD_COMPAT
#include <sys/audit.h>
#include <sys/uio.h>

main()
{
     char buf[100];
     int i;
     struct iovec iov;

     for (i = 0; i < sizeof(buf); i++)
          buf[i] = i;

     iov.iov_len = sizeof(buf);
     iov.iov_base = buf;

     if (audgenl (AUDGEN8,
                  AUD_T_CHARP, "opaque data test",
                  AUD_T_OPAQUE, &iov,
                  0 ) ==  -1 )
         perror ("audgenl");
}
```

19.3.2 Private Tokens

Private tokens are used by the kernel; they are not available to application programs. The `audgen()` system call rejects any attempts by application programs to write records that contain private tokens. Private tokens are defined in `<sys/audit.h>` and begin with `AUD_TP_`; for example `AUD_TP_AUID`.

The kernel uses the private tokens when creating audit records. For example, the kernel encapsulates each audit record with `AUD_TP_LENGTH` tuples whose value is the length of the audit record. Another example is the `audgen()` or `audgenl()` *event* argument, from which the kernel creates a `AUD_TP_EVENT` tuple.

19.4 Audit Flag and Masks

Whether an audit event actually results in the generation of an audit record depends on the following flag and mask settings:

- Process audit control flag
- Process audit mask
- System audit mask

The process audit control flag has four exclusive states:

AUDIT_OR	An audit record is generated if either the system audit mask or the process audit mask indicates such an event should be audited.
AUDIT_AND	An audit record is generated if both the system audit mask and the process audit mask indicate such an event should be audited.
AUDIT_OFF	No audit records are generated for the current process.
AUDIT_USR	An audit record is generated if the process audit mask indicates such an event should be audited.

The process audit control flag also has two nonexclusive states:

AUDIT_SYSCALL_OFF	Turns off system call record generation for the process.
AUDIT_HABITAT_USR	Turns on the habitat system calls in the user mask for the process even if system calls are turned off for the

system mask. The habitat system calls are: System V – `unlink()` and `open()`; real time – `memlk()`, `memunlk()`, `psx4_time_drift()`, and `rt_setprio()`. These habitat system calls are turned on or off as a group. See Appendix B for the habitat events.

The system administrator can establish a default audit level for users, while retaining the ability to audit any individual user at whatever level the administrator deems appropriate. (See Chapter 10 for information on configuring and administering the audit subsystem.)

From a programmer's perspective, a privileged process can set its audit level (specify what gets audited), either as an absolute mask or in relation to the system audit mask. See Section 19.6 for an example showing how to set a process's audit mask. See `audcntl`(2) and `auditmask`(8) for more information.

19.5 Disabling System-Call Auditing for the Current Process

Controlling which events are audited is an important step in fine-tuning the amount of audit data collected. System calls can generate large amounts of audit data, but this data is not necessarily useful information. In general, actively auditing the modification of fields in a security-relevant database or auditing a specific security-relevant action provides more usable information than trying to derive this information from a multitude of system-call audit records. For example, the login process executes thousands of system calls, but a single informative audit record written by the login process uses less system resources and is easier to understand.

Application programs can disable system-call auditing but still allow trusted-event auditing. The following code fragment shows how to use the `audcntl()` system call to set `AUDIT_SYSCALL_OFF`:

```
/* OR the AUDIT_SYSCALL_OFF bit into the audcntl flag */
if ((cntlflag = audcntl(GET_PROC_ACNTL,
                        NULL, 0, 0, 0, 0)) == -1)
    perror("audcntl");
else
    audcntl(SET_PROC_ACNTL, NULL, 0,
            cntlflag|AUDIT_SYSCALL_OFF, 0, 0);
```

19.6 Modifying System-Call Auditing for the Current Process

A process can control what is audited for itself or another process by modifying the target process's `auditmask` and `audcntl` flags. You can modify the current process's audit mask as follows:

```
/* ex. set the process's auditmask to audit only LOGIN
   events and successful setgroups calls
*/
#include <sys/audit.h>
#include <sys/syscall.h>
char buf[AUDIT_MASK_LEN];
.
.
.
bzero (buf, sizeof(buf));
A_PROCMASK_SET (buf, LOGIN, 1, 1);
A_PROCMASK_SET (buf, SYS_setgroups, 1, 0);
if (audcntl (SET_PROC_AMASK, buf,
             AUDIT_MASK_LEN, 0, 0, 0) == -1)
    perror ("audcntl");
```

The A_PROCMASK_SET macro, defined in `<sys/audit.h>`, takes the following arguments:

buf	The buffer containing the mask.
event name	The `<sys/audit.h>` header file contains trusted event names. The `<sys/*syscall.h>` header files contain system call names.
succeed	Indicates whether to audit success; a `1` means audit event success.
fail	Indicates whether to audit failure; a `1` means audit event failure.

See `audcntl(2)` for more information.

19.7 Application-Specific Audit Records

An application program provides application-specific audit data as arguments to `audgen()` or `audgenl()`.

The following code fragment sends an audit record to the kernel when the specified *event* occurs. The *event* is either a trusted event from `<sys/audit.h>` or a site-defined event from `/etc/sec/site_events`.

(Whether the kernel actually writes an audit record to the audit log depends on the events audited for this process.)

```
/* If bad_thing occurs, generate an event of type event_num,
 * with string "bad thing happened", and a result of 66.
 */

#include <sys/audit.h>

if (bad_thing) {
   if (audgenl (event_num,
                AUD_T_CHARP, "bad thing happened",
                AUD_T_RESULT, 66, 0 ) == -1)
      perror ("audgenl");
}
```

In general, an application-generated audit record does not have to include data for the tokens listed in Table 19–1. The kernel automatically adds this information to each audit record. However, the audit subsystem does not prevent you from putting any of the public token tuples in an audit record; for example, you can add an AUD_T_AUID tuple to an audit record even though the system will later add an AUD_TP_AUID to the record. Both tuples are written to the audit log.

19.8 Site-Defined Events

A site can define its own set of audit events, called **site-defined events**, in the locally created and maintained file /etc/sec/site_events. The file contains one entry for each site event.

The potential range for site event numbers is MIN_SITE_EVENT (defined in <sys/audit.h>) to 1048576. The default range is 64. To change this value, set audit-site-events in /etc/sysconfigtab and reboot. For example, to allow for up to 128 site-defined events:

```
sec:
    audit-site-events=128
```

Each site-event entry can contain up to INT_MAX subevents. There is no default range defined for subevents.

The maximum length for an event or subevent name is AUD_MAXEVENT_LEN, defined in <sys/audit.h>.

Application programs can generate records containing both site-defined events and the trusted events defined in <sys/audit.h> (MIN_TRUSTED_EVENT to MAX_TRUSTED_EVENT).

The auditmask utility supports preselection for site-defined events, and the audit_tool utility supposts postselection for site-defined events and subevents.

19.8.1 Sample site_events File

The syntax for a site-defined audit event entry is:

[*event_name event_number* [, *subevent_name subevent_number* ...] ;]

The following entries in a sample /etc/sec/site_events file demonstrate how to create site-defined events and subevents:

```
essence 2048,     1
    ess_read 0,   2
    ess_write 1;  3
rdb 2049,
    rdb_open 0,
    rdb_close 1,
    rdb_read 2,
    rdb_write 3;
decinspect 2050;
```

Notes:

1. essence is the event; 2048 is the event number. Note that 2048 is MIN_SITE_EVENT, the lowest number available for site-defined events.
2. ess_read is the first subevent; 0 is the first subevent number.
3. ess_write is the second subevent; 1 is the second subevent number.

See aud_sitevent(3) for more information on site-defined events.

19.8.2 Example: Generating an Audit Record for a Site-Defined Audit Event

The following code fragment uses audgenl() to generate audit data for an rdb_close event:

```
int event_num, subevent_num;

/* translate event name(s) into event numbers */
if (aud_sitevent_num ("rdb", "rdb_close",
                      &event_num, &subevent_num ))
    printf ("aud_sitevent_num failed");

/* generate audit data */
else if (audgenl (event_num,
                  AUD_T_SUBEVENT, subevent_num,
                  AUD_T_CHARP, "Trusted RDB V1.0 Close",
                  0) == -1)
```

```
perror ("audgenl");
```

Compaq recommends that you include an AUD_T_CHARP, *event name* argument pair with audgenl() when generating a record for a site-defined event. Doing so simplifies the task of analyzing audit data on a system that does not have a copy of the local site_events file.

See aud_sitevent(3) and audgenl(3) for more information.

19.9 Creating Your Own Audit Logs

You can use the audgen() system call to create your own audit log. If the *size* argument to audgen() is a nonzero value, audit data is copied to the *userbuff* specified in audgen() rather than written to the system audit logs. A trusted application can then write the data in *userbuff* to a unique log file. See audgen(2) for more information.

You can use the audit_tool utility to read the new audit log. More detailed information can be read from the log using the information in Section 19.10.

19.10 Parsing an Audit Log

Most people use audit_tool or dxaudit to read audit logs. The audit_tool utility is a sophisticated program that converts audit data into useful information, formats output, and handles audit records that span audit log files. When audit_tool first reads an audit log, it creates a corresponding .hdr file to maintain state information. This state information reduces the time needed for subsequent reads of the audit logs. Also, if an audit record spans audit logs, audit_tool opens both log files and creates a complete record in the header file.

The following sections describe the format and construction of audit logs; they provide:

- A description of an audit log plus a list of the token types generally found in all audit records.
- The binary record format with examples showing an octal dump of a record and its formatted output.
- A table of token/tuple byte descriptions, which lists the data types and format for each public and private token.
- Sample macros for parsing tuples.

These sections do not provide the design information needed to create a program similar to audit_tool; they do provide the basic information required to parse an audit log into records and tuples.

19.10.1 Overview of Audit Log Format and List of Common Tuples

Audit logs are regular UNIX data files that contain audit records. An audit record consists of a series of tuples whose format is either *token:value* or *token:length:value*. Each record starts and ends with an AUD_TP_LENGTH tuple. (The audit_tool utility uses AUD_TP_LENGTH to determine whether an audit record is valid. If the actual length of the record does not match the AUD_TP_LENGTH value, audit_tool discards the record and provides a warning.) Table 19–1 shows the default tuples generally used for audit records.

Table 19–1: Default Tuples Common to Most Audit Records

Tuple	Comment	Tuple	Comment
AUD_TP_LENGTH		AUD_TP_VERSION	
AUD_TP_AUID		AUD_TP_RUID	
AUD_TP_HOSTADDR		AUD_TP_EVENTP	if habitat
AUD_TP_HABITAT	if habitat	AUD_TP_EVENT	
AUD_TP_UID		AUD_TP_PID	
AUD_TP_PPID		AUD_TP_DEV	if device is associated with a process
AUD_TP_NCPU		AUD_TP_TV_USEC	
AUD_TP_SET_UIDS	if uid change	AUD_TP_TID	if AUDIT_USR flag is set

19.10.2 Token/Tuple Byte Descriptions

Table 19–2 lists public and private tokens with their octal values. For each tuple, the third column lists the sequence in which tuple data is written to an audit log by the kernel. Token is a 1–byte quantity; length is a 4–byte quantity. Sample Parse Macro refers to the macro that audit_tool uses to parse the tuple. These macros are provided, for reference purposes only, in Section 19.10.3.

Table 19–2: Token/Tuple Byte Descriptions

Token	Octal Value	Tuple Format and Sample Parse Macro
AUD_T_CHARP	001	token:length:null-terminated ASCII string. PARSE_DEF3
AUD_T_SOCK	003	token:length:struct sockaddr (4.3 style (u_short); if family is > UCHAR_MAX, assume 4.4 style sockaddr of length (byte) then family (byte)). PARSE_DEF3
AUD_T_LOGIN	004	token:length:null-terminated ASCII string. PARSE_DEF3
AUD_T_HOMEDIR	005	token:length:null-terminated ASCII string. PARSE_DEF3
AUD_T_SHELL	006	token:length:null-terminated ASCII string. PARSE_DEF3
AUD_T_DEVNAME	007	token:length:null-terminated ASCII string. PARSE_DEF3
AUD_T_SERVICE	010	token:length:null-terminated ASCII string. (reserved for future use)
AUD_T_HOSTNAME	011	token:length:null-terminated ASCII string. PARSE_DEF3
AUD_T_INTP	012	token:length:int (First element is number of elements in array; note that AUD_T_INTARRAY is the preferred tuple when generating an audit record.) PARSE_DEF3
AUD_T_LSOCK	016	
AUD_T_RSOCK	017	
AUD_T_LHOSTNAME	020	
AUD_T_OPAQUE	030	token:length:value. (proplist or truly opaque; check for proplist name/value pairs else dump as hex) PARSE_DEF6
AUD_T_INTARRAY	031	token:length:int. PARSE_DEF3
AUD_T_GIDSET	032	token:length:int1, int2, ... (unaligned). PARSE_DEF3
AUD_T_XDATA	033	token:struct aud_xdata (See `<sys/audit.h>`.) PARSE_DEF8
AUD_T_AUID	040	token:int. PARSE_DEF2
AUD_T_RUID	041	token:int. PARSE_DEF2
AUD_T_UID	042	token:int. PARSE_DEF2
AUD_T_PID	043	token:int. PARSE_DEF2

Table 19–2: Token/Tuple Byte Descriptions (cont.)

Token	Octal Value	Tuple Format and Sample Parse Macro
AUD_T_PPID	044	token:int. PARSE_DEF2
AUD_T_GID	045	token:unsigned int. PARSE_DEF2
AUD_T_EVENT	046	token:int. PARSE_DEF2
AUD_T_SUBEVENT	047	token:int. PARSE_DEF2
AUD_T_DEV	050	token:int (Parse using the `major()`/`minor()` macros from `<sys/types.h>`.) PARSE_DEF2
AUD_T_ERRNO	051	token:int. PARSE_DEF1
AUD_T_RESULT	052	token:long. PARSE_DEF4
AUD_T_MODE	053	token:unsigned int. PARSE_DEF2
AUD_T_HOSTADDR	054	token:unsigned int. PARSE_DEF2
AUD_T_INT	055	token:int. PARSE_DEF2
AUD_T_DESCRIP	056	token:int (file descriptor). PARSE_DEF2
AUD_T_HOSTID	057	token:int. PARSE_DEF1
AUD_T_X_ATOM	060	token:unsigned int. PARSE_DEF2
AUD_T_X_CLIENT	061	token:int. PARSE_DEF2
AUD_T_X_PROPERTY	062	token:int. PARSE_DEF2
AUD_T_X_RES_CLASS	063	token:unsigned int. PARSE_DEF2
AUD_T_X_RES_TYPE	064	token:unsigned int. PARSE_DEF2
AUD_T_X_RES_ID	065	token:unsigned int. PARSE_DEF2
AUD_T_LHOSTNAME	066	
AUD_T_SECEVENT	177	token:int. PARSE_DEF2
AUD_TP_ACCRGHT	201	token:length:cmsg_data (fd1, fd2, ... – See `<sys/socket.h>`.) PARSE_DEF3
AUD_TP_MSGHDR	202	token:length:msghdr->msg_name. (See `<sys/socket.h>`.) PARSE_DEF3
AUD_TP_EVENTP	203	token:length:string. PARSE_DEF3
AUD_TP_HABITAT	204	token:length:string. PARSE_DEF3
AUD_TP_ADDRVEC	205	token:length:struct sockaddr. (See `socket.h`.) PARSE_DEF3
AUD_TP_INTP	206	token:length:int. PARSE_DEF3
AUD_TP_AUID	241	token:int. PARSE_DEF1

Table 19–2: Token/Tuple Byte Descriptions (cont.)

Token	Octal Value	Tuple Format and Sample Parse Macro
AUD_TP_RUID	0242	token:int. PARSE_DEF1
AUD_TP_UID	0243	token:int. PARSE_DEF1
AUD_TP_PID	0244	token:int. PARSE_DEF1
AUD_TP_PPID	0245	token:int. PARSE_DEF1
AUD_TP_HOSTADDR	246	token:unsigned int. PARSE_DEF1
AUD_TP_EVENT	247	token:int. PARSE_DEF1
AUD_TP_SUBEVENT	250	token:int (Reserved for future use.) PARSE_DEF1
AUD_TP_NCPU	251	token:int. PARSE_DEF1
AUD_TP_DEV	252	token:int (Parse using the `major()`/`minor()` macros from `sys/types.h`.) PARSE_DEF1
AUD_TP_LENGTH	253	token:int. PARSE_DEF1
AUD_TP_IPC_GID	254	token:unsigned int (ipc|msg|shm_perm.gid). PARSE_DEF2
AUD_TP_IPC_MODE	255	token:unsigned int (ipc|msg|shm_perm.mode). PARSE_DEF2
AUD_TP_IPC_UID	256	token:int (ipc|msg|shm_perm.uid). PARSE_DEF2
AUD_TP_TV_SEC	257	token:timeval.tv_sec (See `<sys/time.h>`.) PARSE_DEF1
AUD_TP_TV_USEC	260	token:timeval.tv_usec (See `<sys/time.h>`.) PARSE_DEF1
AUD_TP_SHORT	261	token:short. PARSE_DEF2
AUD_TP_LONG	262	token:long. PARSE_DEF5
AUD_TP_VNODE_DEV	263	token:int. PARSE_DEF2
AUD_TP_VNODE_ID	264	token:unsigned int. PARSE_DEF2
AUD_TP_VN-ODE_MODE	265	token:unsigned int. PARSE_DEF2
AUD_TP_VERSION	266	token:unsigned int. (See `<sys/audit.h>`.) (AUD_VERSION | AUD_VERS_LONG). PARSE_DEF1
AUD_TP_SET_UIDS	267	token:int. PARSE_DEF2
AUD_TP_CONT	270	token:unsigned int (A unique int for each component of a record.) PARSE_DEF1

Table 19–2: Token/Tuple Byte Descriptions (cont.)

Token	Octal Value	Tuple Format and Sample Parse Macro
AUD_TP_TID	271	token:long. PARSE_DEF4
AUD_TP_PRIV	272	token:unsigned short. PARSE_DEF1

19.10.3 Parsing Tuples

The algorithm for reading a stream of audit records is as follows:

1. Open the audit log.
2. Find the first audit record (starts and ends with AUD_TP_LENGTH tuples).
3. Check that the record length matches the value in the AUD_TP_LENGTH tuple. (If the length does not match, discard the record.)
4. Retrieve the first tuple following the AUD_TP_LENGTH tuple.
5. If the tuple length is variable, determine the size of the data.
6. Extract the data.
7. Retrieve the next tuple, check the length if necessary, and extract the data.
8. Repeat until no more records.
9. Close the audit log.

The following macros illustrate how audit_tool parses tuples. The macros are provided for reference purposes only; they illustrate one approach. Note that *indx* values are maintained and used by audit_tool; they are not part of the audit record tuple.

```
/* fixed length scalar value */
#define PARSE_DEF1(tokentype,field) \
  bcopy (&rec_ptr[i+sizeof token], &field, sizeof(field)); \
  i += (sizeof token + sizeof(field)); \
  break;

/* fixed length field in array */
#define PARSE_DEF2(tokentype,field,indx) \
  if (indx < AUD_NPARAM) \
      bcopy (&rec_ptr[i+sizeof token], &field[indx++], sizeof(field[0])); \
  i += (sizeof token + sizeof(field[0])); \
  break;

/* array of strings */
#define PARSE_DEF3(tokentype,len,field,indx) \
  bcopy (&rec_ptr[i+sizeof token], &j, sizeof(int)); \
  if (j >= rec_len) j = 0; \
  if (indx < AUD_NPARAM) { \
      len[indx] = j; \
      field[indx++] = (char *)&rec_ptr[i+(sizeof token)+(sizeof *intp)]; \
```

```
  } \
  i += (sizeof token + sizeof *intp + j); \
  break;

/* fixed length scalar value whose size is h/w dependent (32 or 64-bit) */
#define PARSE_DEF4(tokentype,field) \
  bzero (field.val, sizeof(field.val)); \
  j = af->version & AUD_VERS_LONG ? sizeof(int)*2 : sizeof(int); \
  bcopy (&rec_ptr[i+sizeof token], field.val, j); \
  i += (sizeof token + j); \
  break;

/* fixed length field in array whose size is h/w dependent (32 or 64-bit) */
#define PARSE_DEF5(tokentype,field,indx) \
  bzero (field[indx].val, sizeof(field[indx].val)); \
  j = af->version & AUD_VERS_LONG ? sizeof(int)*2 : sizeof(int); \
  if (indx < AUD_NPARAM) \
      bcopy (&rec_ptr[i+sizeof token], field[indx++].val, j); \
  i += (sizeof token + j); \
  break;

/* array of opaque data streams */
#define PARSE_DEF6 PARSE_DEF3

/* iovec element in array */
#define PARSE_DEF7(tokentype,field,indx) \
  j = sizeof(field[0]); \
  if (indx < AUD_NPARAM) { \
      bcopy (&rec_ptr[i+sizeof token], &j, sizeof(int)); \
      if (j > rec_len ) j = 0; \
      bcopy (&rec_ptr[i+sizeof token+sizeof(int)], &field[indx++], j); \
  } \
  i += (sizeof token + sizeof(int) + j); \
  break;

/* array of iovec elements with variable length components */
#define PARSE_DEF8(tokentype,field,ptr,indx) \
  j = sizeof(field[0]); \
  if (indx < AUD_NPARAM) { \
      bcopy (&rec_ptr[i+sizeof token], &j, sizeof(int)); \
      if (j > rec_len) j = 0; \
      bcopy (&rec_ptr[i+sizeof token+sizeof(int)], &field[indx], j); \
      ptr[indx++] = ((struct aud_xdata *) \
                     &rec_ptr[i+sizeof token+sizeof(int)])->xdata; \
  } \
  i += (sizeof token + sizeof(int) + j); \
  break;
```

20

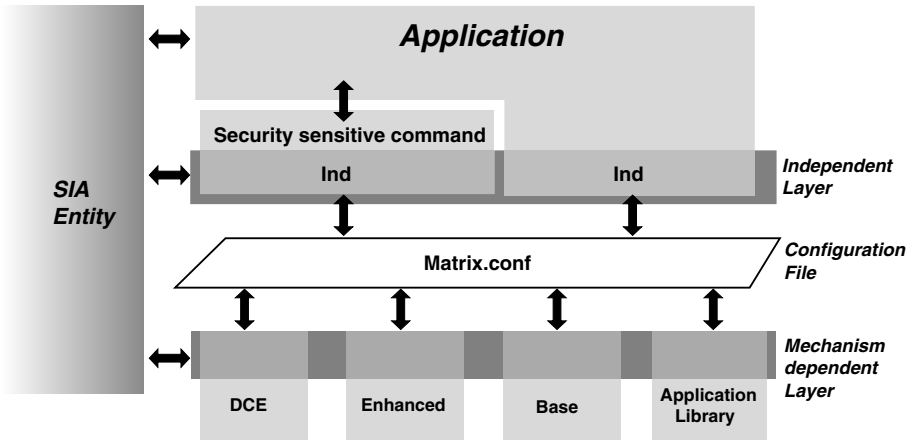
Using the SIA Interface

This chapter documents the Security Integration Architecture (SIA) interfaces.

20.1 Overview

The Security Integration Architecture (SIA) allows the layering of local and distributed security authentication mechanisms onto the Tru64 UNIX operating system. The SIA configuration framework isolates security-sensitive commands from the specific security mechanisms. The Tru64 UNIX security-sensitive commands have been modified to call a set of mechanism-dependent routines. By providing a library with a unique set of routines, developers can change the behavior of security-sensitive commands, without changing the commands themselves. The SIA defines the security mechanism-dependent interfaces (`siad_*()` routines) required for SIA configurability. Figure 20–1 illustrates the relationship of the components that make up the SIA.

Figure 20–1: SIA Layering



ZK–1086U–AI

The security-sensitive commands are listed in Table 20–1.

Table 20–1: Security-Sensitive Operating System Commands

Command	Description
chfn	Changes finger information
chsh	Changes login shell information
dnascd	Spans DECnet
ftpd	Serves the Internet File Transfer Protocol
login	Authenticates users
passwd	Creates or changes user passwords
rshd	Serves remote execution
su	Substitutes a user ID

Table 20–2 and Table 20–3 list the SIA porting routines.

Table 20–2: SIA Mechanism-Independent Routines

SIA Routine	Description
sia_init()	Initializes the SIA configuration
sia_chk_invoker()	Checks the calling application for privileges
sia_collect_trm()	Collects parameters
sia_chg_finger()	Changes finger information
sia_chg_password()	Changes the user's password
sia_chg_shell()	Changes the login shell
sia_ses_init()	Initializes SIA session processing
sia_ses_authent()	Authenticates an entity
sia_ses_reauthent()	Revalidates a user's password
sia_ses_suauthent()	Processes the su command
sia_ses_estab()	Establishes the context for a session
sia_ses_launch()	Logs session startup and any TTY conditioning
sia_ses_release()	Releases resources associated with session
sia_make_entity_pwd()	Provides the password structure for SIAENTITY
sia_audit()	Generates the audit records
sia_chdir()	Changes the current directory safely (NFS-safe)
sia_timed_action()	Calls with a time limit and signal protection
sia_become_user()	su routine

Table 20–2: SIA Mechanism-Independent Routines (cont.)

SIA Routine	Description
sia_validate_user()	Validate a user's password
sia_get_groups()	Gets groups

Table 20–3: SIA Mechanism-Dependent Routines

SIA Routine	Description
siad_init()	Initializes processing once per reboot
siad_chk_invoker()	Verifies the calling program privileges
siad_ses_init()	Initializes the session
siad_ses_authent()	Authenticates the session
siad_ses_estab()	Checks resources and licensing
siad_ses_launch()	Logs the session startup
siad_ses_suauthent()	Processes the su command
siad_ses_reauthent()	Revalidates a user's password
siad_ses_release()	Releases session resources
siad_chg_finger()	Processes the chfn command
siad_chg_password()	Invokes a function to change passwords
siad_chg_shell()	Processes the chsh command
siad_getpwent()	Processes getpwent() and getpwent_r()
siad_getpwuid()	Processes getpwuid() and getpwuid_r()
siad_getpwnam()	Processes getpwnam() and getpwnam_r()
siad_setpwent()	Initializes a series of getpwent() calls
siad_endpwent()	Releases resources after a series of getpwent() calls
siad_getgrent()	Processes getgrent() and getgrent_r()
siad_getgrgid()	Processes getgrgid() and getgrgid_r()
siad_getgrnam()	Processes getgrnam() and getgrnam_r()
siad_setgrent()	Initializes a series of getgrent() calls
siad_endgrent()	Closes series of getgrent() calls
siad_chk_user()	Determines if a mechanism can change the requested information
siad_get_groups()	Fills in the array of a user's supplementary groups

The SIA establishes a layer between the security-sensitive commands and the security mechanisms that deliver the security mechanism-dependent functions. Each of the security-dependent SIA routines can be configured to use up to four security mechanisms, called in varying orders.

The selection and order of the calls to the different security mechanisms is established by a switch table file, `/etc/sia/matrix.conf` (see Chapter 13), similar to the way `/etc/svc.conf` is used to control `libc get*` functions. However, the calling mechanism is distinctly different.

The SIA calling mechanism looks up the addresses of routines in the shared libraries and calls them to access the specific security mechanism routine. SIA provides alternative control and configuration for the `getpw*` and `getgr*` functions in Tru64 UNIX.

SIA layering establishes internationalized message catalog support and thread-safe porting interfaces for new security mechanisms and new security-sensitive commands that need transparency. The thread safety is provided by a set of locks pertaining to types of SIA interfaces. However, because SIA is a layer between utilities and security mechanisms, it is the responsibility of the layered security mechanisms to provide reentrancy in their implementations.

The primary focus for SIA is to provide transparent interfaces for security-sensitive commands like `login`, `su`, and `passwd` that are sufficiently flexible and extensible to suit future security requirements. Any layered product on Tru64 UNIX that is either creating a new security mechanism or includes security-sensitive commands requires SIA integration to preserve these transparent interfaces.

The SIA components consist of only user-level modules. The components resolve the configuration issues with respect to the security-sensitive command's utilization of multiple security mechanisms. The SIA components do not resolve any kernel issues pertaining to the configuration and utilization of multiple security mechanisms.

20.2 SIA Layering

The layering introduced by SIA in Tru64 UNIX consists of the following two groups of interface routines:

`sia_*()`	The security mechanism-independent interface used by security-sensitive commands.
`siad_*()`	The security mechanism-dependent interface supplied by each specific security mechanism.

Each security mechanism delivers a shared library containing the siad_*() routines and provides a unique security mechanism name to satisfy the configuration. The one word security mechanism name and the library name are used as keys in the matrix.conf file to specify which mechanisms to call and in what order.

The Tru64 UNIX security-sensitive commands have been modified to use the mechanism-independent sia_*() routines. These routines are used by the commands and utilities to access security functions yet remain isolated from the specific security technologies. Each sia_*() routine calls the associated mechanism-dependent siad_*() routines, depending on the selected configuration specified in the matrix.conf file. See Chapter 13 for a more detailed discussion of the file.

The mechanism-dependent siad_*() interface routines are defined by SIA as callouts to security mechanism-dependent functions provided by the security mechanisms. The matrix.conf file is used to determine which security mechanisms are called and in what order they are called for each SIA function.

The process of calling a particular module within a specified security mechanism and passing the required state is done by the mechanism-independent layer. The calling process uses shared library functions to access and look up specific module addresses within specified shared libraries provided by the security mechanisms.

The naming of the security mechanism-dependent modules, siad_*() routines, is fixed to alleviate name conflicts and to simplify the calling sequence. Tru64 UNIX uses the dlopen() and dlsym() shared library interfaces to open the specified security-mechanism shared library and look up the siad_*() function addresses. If you need to preempt the siad_*() routines, your names must be of the form __siad_* in your library and the library must be linked ahead of libc. See Appendix D for more information on the naming and preempting requirements.

20.3 System Initialization

The SIA provides a callout to each security mechanism on each reboot of the system. This callout is performed by the /usr/sbin/siainit program, which calls each of the configured security mechanisms at their siad_init() entry point. This allows the security mechanisms to perform a reboot initialization. A SIADFAIL response from the siad_init() call causes the system to not reboot and an SIA INITIALIZATION FAILURE message to be sent to the console. Consequently, only problems that would cause a security risk or would not allow root to log in should warrant a SIADFAIL response from the siad_init() call.

20.4 Libraries

SIA security mechanisms are configured as separate shared libraries with entry points that are SIA defined names. Each mechanism is required to have a unique mechanism identifier. The actual entry points in the shared library provided by the security mechanism are the same for each mechanism, `siad_*()` form entry points.

The default security configuration is the BASE security mechanism contained in `libc`. The default BASE security mechanism uses the `/etc/passwd` file, or a hashed database version, as the user database and the `/etc/group` file as the group's database. The default BASE mechanism also uses the Network Information Service (NIS) if it is configured. In single-user mode or during installation, the BASE security mechanism is in effect.

20.5 Header Files

The SIA interfaces and structures are defined in the `/usr/include/sia.h` and `/usr/include/siad.h` files. The `sia*.h` files are part of the program development subsets.

20.6 SIAENTITY Structure

The SIAENTITY structure contains session processing parameters and is used to transfer session state between the session processing stages. Example 20–1 shows the SIAENTITY structure.

Example 20–1: The SIAENTITY Structure

```
typedef struct siaentity {
    char *name;             /* collected name                */
    char *password;         /* entered or collected password */
    char *acctname;         /* verified account name         */
    char **argv;            /* calling command argument list */
    int  argc;              /* number of arguments           */
    uid_t  suid;            /* starting ruid                 */
    char *hostname;         /* requesting host NULL=>local   */
    char *tty;              /* pathname of local tty         */
    int can_collect_input; /* 1 => yes, 0 => no input       */
    int error;              /* error message value           */
    int authcount;          /* Number of consecutive         */
                            /* failed authent attempts       */
    int authtype;           /* Type of last authent          */
    struct passwd *pwd;     /* pointer to passwd struct      */
    char *gssapi;           /* for gss_api prototyping       */
    char *sia_pp;           /* for passport prototyping      */
    int *mech[SIASWMAX];    /* pointers to mech-specific data  */
```

Example 20–1: The SIAENTITY Structure (cont.)

```
                          /* allocated by mechanisms indexed */
                          /* by the mechind argument         */
} SIAENTITY;
```

20.7 Parameter Collection

The SIA provides parameter collection callback capability so that any graphical user interface (GUI) can provide a callback. The `sia_collect_trm()` routine is used for terminal parameter collection. Commands calling the `sia_*()` routines pass as an argument to the appropriate collection routine pointer, thus allowing the security mechanism to prompt the user for specific input. If the collection routine argument is NULL, the security mechanism assumes that no collection is allowed and that the other arguments must be used to satisfy the request. The NULL case is used for noninteractive commands. For reliability, use a collection routine whenever possible.

The `can_collect_input` argument is included in the session processing and disables the collection facility for input while allowing the output of warnings or error messages. Collection routines support simple form and menu data collection. Some field verification is supported to check parameter lengths and content (alphanumeric, numeric only, letters only, and invisible). The collection routine supplied by the security-sensitive command or utility is responsible for providing the appropriate display characteristics.

The parameter collection capability provided by SIA uses the `sia_collect_trm()`interface which is defined in `sia.h`. See Example 20–2.

Example 20–2: The sia.h Definition for Parameter Collection

```
int sia_collect_trm(timeout, rendition, title,
     num_prompts, prompts);

 int timeout               /* number of seconds to wait */
                           /* 0 => wait forever */
 int rendition
    SIAMENUONE      1      /* select one of the choices given */
    SIAMENUANY      2      /* select any of the choices given */
    SIAFORM         3      /* fill out the form               */
    SIAONELINER     4      /* One question with one answer    */
    SIAINFO         5      /* Information only                */
    SIAWARNING      6      /* ERROR or WARNING message        */

  unsigned char *title      /* pointer to a title string. */
                           /* NULL => no title */
```

Example 20–2: The sia.h Definition for Parameter Collection (cont.)

```
int num_prompts           /* Number of prompts in collection */
prompt_t *prompts         /* pointer to prompts */

typedef struct prompt_t
  {
  unsigned char *prompt;
  unsigned char *result;
  int max_result_length; /* in chars */
  int min_result_length; /* in chars */
  int control_flags;
  } prompt_t;

control_flags
 SIARESINVIS  0x2   result is invisible
 SIARESANY    0x10  result can contain any ASCII chars
 SIAPRINTABLE 0x20  result can contain only printable chars
 SIAALPHA     0x40  result can contain only letters
 SIANUMBER    0x80  result can contain only numbers
 SIAALPHANUM  0x100 result can contain only letters and numbers
```

See the `sia_collect_trm`(3) reference page for more information on parameter collection.

20.8 Maintaining State

Some commands require making multiple calls to `sia_*()` routines and maintaining state across those calls. The state is always associated with a particular user (also called an entity). SIA uses the term entity to mean a user, program, or system which can be authenticated. The entity identifier is the user ID (UID). All security mechanisms which are ported to Tru64 UNIX must be administered such that a particular UID maps equivalently across each mechanism. This constraint allows for the interaction and coexistence of multiple security mechanisms. If a security mechanism has an alternative identifier for a user, it must provide a mapping to a unique UID for other mechanisms to properly interoperate and provide synchronized security information.

A pointer to the SIAENTITY structure (see Section 20.6) is used as an argument containing intermediate state identifying the entity requesting a security session function. The SIAENTITY structure also allows for the sharing of state between security mechanisms while processing a session.

The `libc` library provides for the allocating and freeing of primitives for SIAENTITY structures. The allocation of the SIAENTITY structures

occurs as part of the session initialization routine, sia_ses_init(). The deallocation of the SIAENTITY structure occurs in the call to the session release sia_ses_release() routine. If errors occur during session processing (such as in the sia_ses_*authent() routines) and you give up instead of retrying, sia_ses_release() must be called to clean or free up the SIAENTITY structure related to the session. If errors occur during an sia_ses_estab() or sia_ses_launch() routine causing failure status to be returned, the routines call sia_ses_release().

20.9 Return Values

SIA supports the passing of a success or failure response back to the calling command or utility. The SIAENTITY structure has a reserved error code field (*error*), which is available for finer error definition.

The siad_ses_*() routines return bitmapped values that indicate the following status:

SIADFAIL	Indicates conditional failure. Lowest bit set to 0. Continue to call subsequent security mechanisms.
SIADSUCCESS	Indicates conditional success. Lowest bit set to 1.
SIADSTOP	Modifies the return to be unconditional. Second lowest bit set to 1. Included with either SIADFAIL or SIADSUCCESS.

20.10 Debugging and Logging

SIA supports a debugging and logging capability that allows appending data to the /var/adm/sialog file. The SIA logging facility supports the following three log-item types:

EVENT	Success cases within the SIA processing
ERROR	Failures within the SIA processing
ALERT	Security configuration or security risks within the SIA interfaces

The sia_log() logging routine is available to security mechanisms and accepts formatting strings compatible to printf() format. Each log entry is time stamped. Example 20–3 is a typical /var/adm/sialog file.

Example 20–3: Typical /var/adm/sialog File

```
SIA:EVENT Wed Feb  3 05:21:31 1999
Successful SIA initialization
SIA:EVENT Wed Feb  3 05:22:08 1999
Successful session authentication for terry on :0
SIA:EVENT Wed Feb  3 05:22:08 1999
Successful establishment of session
SIA:ERROR Wed Feb  3 05:22:47 1999
Failure to authenticate session for root on :0
SIA:ERROR Wed Feb  3 05:22:52 1999
Failure to authenticate session for root on :0
SIA:EVENT Wed Feb  3 05:22:59 1999
Successful session authentication for root on :0
SIA:EVENT Wed Feb  3 05:22:59 1999
Successful establishment of session
SIA:EVENT Wed Feb  3 05:23:00 1999
Successful launching of session
SIA:EVENT Wed Feb  3 05:24:40 1999
Successful authentication for su from root to terry
SIA:EVENT Wed Feb  3 05:25:46 1999
Successful password change for terry
```

The `sia_log()` routine is for debugging only. The `_ses_*` routines use `audgen()` for audit logging.

20.11 Integrating Security Mechanisms

Depending on the class or type of SIA processing being requested, the selection and order of security mechanisms may vary. A typical set of security mechanisms might include a local mechanism (one that is only concerned with the local system security) and a distributed security mechanism (one that is concerned with aspects of security that span several systems). SIA layering allows these two security mechanisms to either coexist or be better integrated.

An example of security mechanism integration is the log in or session processing. SIA layering passes state (SIAENTITY) between the various security mechanisms during the session processing. This state contains collected names and passwords and the current state of session processing. The local security mechanism can be designed to trust the authentication process of a previously run security mechanism, thus allowing authentication vouching. In this case, if a user is successfully authenticated by the distributed mechanism, the local mechanism can accept or trust that authentication and continue with session processing.

SIA also allows the local mechanism to not accept vouching. In this case, the local mechanism would be forced to do its own authentication process regardless of previous authentication outcomes. This typically results in the user being asked for several sets of user names and passwords. Although SIA allows any ordering of security mechanisms, it makes sense that those mechanisms that accept vouching should be ordered after those that do not.

Notes

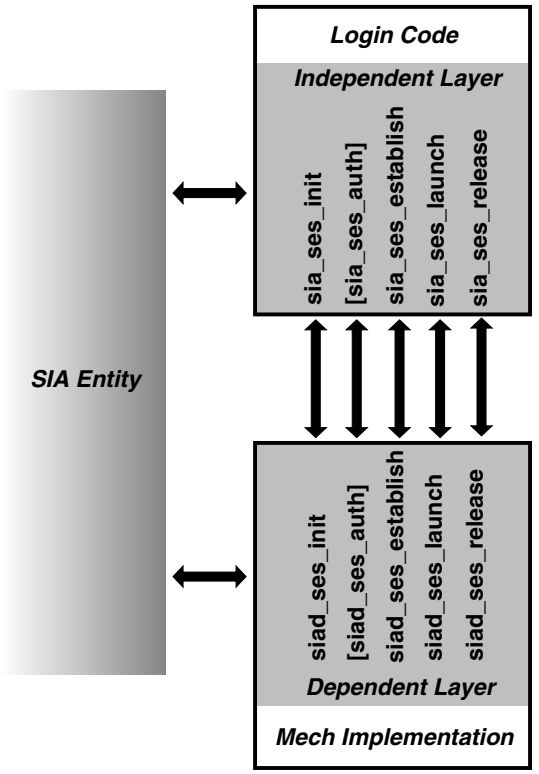
The default security mechanism, BASE, accepts authentication vouching.

The SIA layer deals with the isolation of security mechanisms from the commands' specific user interface preferences. To accomplish this isolation, the calling command provides a pointer to a parameter collection routine as an argument to the `sia_*()` routines. The collection routine must support simple form and menu type processing. The definitions or the requirements of the collection routine are defined in `sia.h`. This separation of user interface from the security mechanisms allows the flexibility to change the user interface to suit any workstation or dumb terminal model.

20.12 Session Processing

The session processing interfaces are associated with the process of a utility or command that needs to become or act as some other entity. Figure 20–2 illustrates the SIA routines and their relationship in a typical login session.

Figure 20–2: SIA Login Session Processing



ZK–1085U–AI

The session processing interfaces to the security mechanism-dependent routines (`siad_*()`) use the same returns to determine the state of the session and whether it should continue. The returns are as follows:

SIADFAIL	A SIADFAIL response from a security mechanism `siad_*()` routine indicates that the security mechanism has failed but that processing should continue.
SIADFAIL | SIASTOP	A SIADFAIL | SIADSTOP response from a security mechanism `siad_*()` routine indicates that the security mechanism has failed and that the session processing should be stopped. This return is used if some major security problem or risk is found. Such an event should be sent to the `sialog` file as an ALERT.

SIADSUCCESS	The final response is SIADSUCCESS, which indicates that the security mechanism has successfully completed that phase of session processing. Under some conditions, a return of SIADSUCCESS | SIADSTOP is also useful.

Not all security mechanisms have processing required in each phase of the session processing. In general, the default response is SIADFAIL to force the other configured security mechanisms to produce the required SIADSUCCESS response. The only exceptions to this are the first and last stages of session processing. If a security mechanism has nothing to do in either session initialization or session release, it should return a SIADSUCCESS response. For all other phases of session processing, a SIADFAIL response is the default.

The session processing interfaces are typically called in the following order:

`sia_ses_init()`	Initialize the session.
`sia_ses_authent()`	Authenticate the session. Can be recalled on failure for retries.
`sia_ses_estab()`	Establish the session. On failure, calls `sia_ses_release()`.
`sia_ses_launch()`	Launch the session. On failure, calls `sia_ses_release()`.
`sia_ses_release()`	Release the session.

The session routines must all have the same number and order of mechanisms to keep the mechanism index (`mechind`) consistent.

Example 20–4 is a code fragment that shows session processing for the `login` command.

Example 20–4: Session Processing Code for the login Command

```
  .
  .
  .
/* SIA LOGIN PROCESS BEGINS */

/* Logging of failures to sia_log is done within the libsia */
/* Logging to syslog is responsibility of calling routine   */
```

Example 20–4: Session Processing Code for the login Command (cont.)

```
if((sia_ses_init(&entity, oargc, oargv, hostname, loginname, \
   ttyn, 1, NULL)) == SIASUCCESS) {

/***** SIA SESSION AUTHENTICATION *****/

   if(!fflag) {
      for(cnt=5; cnt; cnt--) {
         if((authret=sia_ses_authent(sia_collect,NULL,entity)) \
   == SIASUCCESS)
           break;
         else    if(authret & SIASTOP)
           break;
         fputs(MSGSTR(INCORRECT, "Login incorrect\n"), stderr);
      }
      if(cnt <= 0 || (authret & SIASTOP)) {
         sia_ses_release(&entity);
         exit(1);
      }
   }

/***** SIA SESSION ESTABLISHMENT *****/

   if(sia_ses_estab(sia_collect,entity) == SIASUCCESS) {
      /****** set up environment   *******/
      /* destroy environ. unless user requested preservation */
      if (!pflag) {
        pp = getenv("TERM");
        if (pp)
        strncpy(term, pp, sizeof term);
        clearenv();
   }
   (void)setenv("HOME", entity->pwd->pw_dir, 1);
      if(entity->pwd->pw_shell && *entity->pwd->pw_shell)
        strncpy(shell, entity->pwd->pw_shell, sizeof shell);
   (void)setenv("SHELL", shell, 1);
   if (term[0] == ' ')
      (void)strncpy(term, stypeof(tty), sizeof(term));
   (void)setenv("TERM", term, 0);
   (void)setenv("USER", entity->pwd->pw_name, 1);
   (void)setenv("LOGNAME", entity->pwd->pw_name, 1);
   (void)setenv("PATH", _PATH_DEFPATH, 0);

/***** SIA LAUNCHING SESSION *****/

   if(sia_ses_launch(sia_collect,entity) == SIASUCCESS) {
   /* 004 - start */
   if ((entity -> pwd           != NULL) &&
```

Example 20–4: Session Processing Code for the login Command (cont.)

```
       (entity -> pwd -> pw_dir != NULL) &&
       (entity -> pwd -> pw_dir [0] != 0))
          sprintf (hush_path, "%s/%s",
             entity -> pwd -> pw_dir,
             _PATH_HUSHLOGIN);
   else    strcpy (hush_path, _PATH_HUSHLOGIN);
   quietlog = access(hush_path, F_OK) == 0;
   /* 004 - end */
   if(!quietlog)
      quietlog = !*entity->pwd->pw_passwd && \
   !usershell(entity->pwd->pw_shell);
         if (!quietlog) {
            struct stat st;
            motd();
            (void)sprintf(tbuf, "%s/%s", _PATH_MAILDIR, \
    entity->pwd->pw_name);
            if (stat(tbuf, &st) == 0 && st.st_size != 0)
               (void)printf(MSGSTR(MAIL, "You have %smail.\n"),
                     (st.st_mtime > st.st_atime) ? MSGSTR(NEW, \
     "new ") : );
         }
         sia_ses_release(&entity);

/******* Setup default signals **********/

         (void)signal(SIGALRM, SIG_DFL);
         (void)signal(SIGQUIT, SIG_DFL);
         (void)signal(SIGINT, SIG_DFL);
         (void)signal(SIGTSTP, SIG_IGN);

         tbuf[0] = '-';
         (void)strcpy(tbuf + 1, (p = rindex(shell, '/')) ?
                        p + 1 : shell);

/****** Nothing left to fail *******/

         if(setreuid(geteuid(),geteuid()) < 0) {
            perror("setreuid()");
            exit(3);
         }
         execlp(shell, tbuf, 0);
         (void)fprintf(stderr, MSGSTR(NO_SHELL, \
  "login: no shell: %s.\n"), strerror(errno));
         exit(0);
      }
/***** SIA session launch failure *****/
    }
```

Example 20–4: Session Processing Code for the login Command (cont.)

```
/***** SIA session establishment failure *****/
  }
  logerror(entity);
  exit(1);
}

logerror(entity)
SIAENTITY *entity;
{
  if(entity != NULL)
    {
    sia_ses_release(&entity);
    }
  syslog(LOG_ERR, MSGSTR(FAILURE3," LOGIN FAILURE "));
}
  .
  .
  .
```

20.12.1 Session Initialization

Session initialization is performed by the `sia_ses_init()` routine. The `sia_ses_init()` routine calls each configured security mechanism's `siad_ses_init()` entry point to do any processing associated with the start of a session processing sequence. The session initialization stage is responsible for setting up the SIAENTITY structure, which is used to maintain state though the different stages of session processing.

20.12.2 Session Authentication

The authentication stage of session processing is responsible for proving the identity for the session. This stage of the processing must determine the entity associated with the session. If the entity cannot be determined, the authentication fails. If the authentication is successful, an entity is derived.

The top level SIA session authentication routine, `sia_ses_authent()`, calls the security mechanism-dependent `siad_ses_authent()` routines according to the configured sequence stored in the `matrix.conf` file. As the multiple authentication routines are called, the SIAENTITY structure is used to hold precollected parameters like the name, password, and eventually the associated `/etc/passwd` entry of the entity.

By using precollected arguments, the security mechanisms avoid recollecting arguments. An example is when `root` attempts to log in to a system

configured to first call the DCE `siad_ses_authent()` routine followed by the local ENHANCED (enhanced security) `siad_ses_authent()` routine.

It is likely that the DCE authentication process will not be capable of authenticating `root`. However, it is capable of asking the user for a name and password, which are then passed to the ENHANCED `siad_ses_authent()` routine using the SIAENTITY structure. This allows the ENHANCED mechanism to verify the `root` name and password, thus authenticating `root`. As soon as the session authentication stage is complete, the password field is cleared.

Each security mechanism-dependent authentication routine must have the ability to determine and set the entity on a successful authentication. If a security mechanism has its own private interpretation of the entity, it must provide a translation to the common SIA entity, user name and UID. Without this restriction there is no way to synchronize security mechanisms with respect to a common entity.

At the successful completion of the session authentication stage, the SIAENTITY structure must contain the user name and UID of the authenticated entity. If the session authentication fails, the calling command or program can call `sia_ses_authent()` again to retry the authentication process. Certain mechanisms may allow other mechanisms to vouch for this stage of session processing. This usually occurs when local mechanisms default their authentication process to other distributed mechanisms.

20.12.3 Session Establishment

The session establishment stage is invoked with `sia_ses_estab()` following a successful session authentication stage. The `sia_ses_estab()` routine is configured to call multiple security mechanism's `siad_ses_estab()` routines in the order defined in the `matrix.conf` file. The session establishment stage of session processing is responsible for checking mechanism resources and licensing to determine whether this session can be successfully launched. The determination of the `passwd struct` entry and any other required security context must occur in this stage. At the successful completion of the session establishment stage, the system is prepared to grant the session launching.

20.12.4 Session Launch

The session launch stage is responsible for the logging and the accounting of the session startup. The local mechanism is additionally responsible for setting the `wtmp` and `utmp` entries and for setting the effective UID to the UID associated with the entity. The processing by the `setgid()` and `initgroup()` routines as well as `lastlog` updating are also done by the

local mechanism. Only catastrophic errors should be able to stop the session from continuing.

20.12.5 Session Release

The last stage of the session processing sequence (either successful or failed) is the call to the `sia_ses_release()` routine. This routine frees all session processing resources, such as the SIAENTITY structure. Each configured mechanism is called to release any resources which are no longer required for the session.

20.12.6 Specific Session Processing

The following sections describe specific session processing for the `login`, `rshd`, and `rlogind` commands. See Section 20.12 for a generic description of session processing.

20.12.6.1 The login Process

The most common case of session processing is when the login process becomes the entity associated with a user. The entity is the unique SIA identifier for any person or process that can be authenticated and authorized. The code in Example 20–4 is part of the `login` command.

20.12.6.2 The rshd Process

Session processing for `/usr/sbin/rshd` differs from `login`. The `rshd` process calls `ruserok()` to check the `.rhosts` and `host.equiv` files for authorization. If `ruserok()` fails, the `rshd` fails.

20.12.6.3 The rlogind Process

The `rlogind`, program executes the `login` command with the `–f` flag if its call to `ruserok()` is successful, and without the `–f` flag if the call to `ruserok()` is unsuccessful. If `login` is executed without the `–f` flag, `sia_ses_authent()` is called, which prompts for a user name and password, if required.

20.13 Changing Secure Information

The routines described in this section handle the changing of the traditional `/etc/passwd` entry information. This class of routines could be extended to handle other types of common secure information. Only the traditional `passwd`, `chfn`, and `chsh` types of command processing are specified. Each of these routines follows the same operational model. When a user requests a change, the routines in this class check each mechanism that was configured by calling `siad_chk_user()` to determine whether the user is registered

with the mechanism. Once it is determined that the user is registered with more than one security mechanism, the user is given a menu selection by the collection routine to choose which mechanism is targeted for the change. If only one mechanism is configured to handle the request, then that mechanism is called directly.

20.13.1 Changing a User's Password

To change a password, the `sia_chg_password()` routine calls the configured mechanisms by using the `siad_chg_password()` routine. To determine which mechanisms support a particular user, the `siad_chk_user()` call is made to all mechanisms configured for the `siad_chg_passwd()` routine. When multiple mechanisms claim registry of a user, the user is given a selection to choose from. If the user is only registered with one mechanism, then that mechanism is called.

20.13.2 Changing a User's Finger Information

The `sia_chg_finger()` routine calls the configured mechanisms by the `siad_chg_finger()` routine to change finger information. To determine which mechanisms support a particular user, the `siad_chk_user()` call is made to all mechanisms configured for the `siad_chg_finger()` routine. When multiple mechanisms claim registry of the user, the user is given a selection menu to choose one from. If the user is only registered with one mechanism, then that mechanism is called.

20.13.3 Changing a User's Shell

The `sia_chg_shell()` routine calls the configured mechanisms by the `siad_chg_shell()` routine to change a user's login shell. To determine which mechanisms support a particular user, the `siad_chk_user()` call is made to all mechanisms configured for the `siad_chg_shell()` routine. When multiple mechanisms claim registry of the user, the user is given a selection menu from which to choose a mechanism. If the user is only registered with one mechanism, then that mechanism is called.

20.14 Accessing Security Information

The SIA interfaces described in the following sections handle the access to the traditional UNIX `/etc/passwd` and `/etc/group` information. You can create routines to handle the access of other common secure information. Mechanism-dependent security information access should not be handled by the SIA interfaces unless nearly all mechanisms support the type of information being accessed.

The sia_context and mech_contexts structures, defined in sia.h, are used to maintain state across mechanisms. The structures are as follows:

```
struct mech_contexts {
        void *value;
        void (*destructor)();
};

struct sia_context {
        FILE *fp;
        union {
        struct group *group;
        struct passwd *pass;
        } value;
        int pkgind;
        unsigned buflen;
        char *buffer;
        struct mech_contexts mech_contexts[SIASWMAX];
};
```

Because the getgr*() and the getpw*() routines have SIA interfaces, security mechanisms need provide only one routine for both reentrant and nonreentrant, threadsafe applications. This is accomplished by the sia_getpasswd() and sia_getgroup() routines which encapsulate the arguments in a common form for the security mechanism's siad_*() routines.

20.14.1 Accessing /etc/passwd Information

Access to traditional /etc/passwd entries is accomplished by the getpw*() routines in libc and libc_r. The sia_getpasswd() routine in the SIA layer preserves the calling semantics of the current getpw*() routines and converts them into one common routine used for both single and multithreaded processes. By doing this conversion, security mechanisms need only support one set of getpw*() routines. The processing of the getpwent() routine is accomplished by calling each configured security mechanism in the predefined order until all entries have been exhausted.

20.14.2 Accessing /etc/group Information

Access to traditional /etc/group entries is accomplished by the getgr*() routines in libc and libc_r. The sia_getgroup() routine in the SIA layer preserves the calling semantics of the current getgr*() routines and converts them into one common routine used for both single and multithreaded processes. The conversion to a single routine eases the security mechanism port by reducing the number of routines required. The processing of the getgrent() routine is accomplished by calling each

configured security mechanism in the predefined order until all group entries have been exhausted.

20.15 Session Parameter Collection

The SIA session interfaces and the interfaces that change secure information use a predefined parameter collection capability. The calling application passes the address to a parameter collection routine through the SIA to the `siad_*()` routines. The collection routine allows different security mechanisms to prompt the user for different parameters without having to be aware of the user interface details.

This capability isolates the SIA security mechanisms from the user interface and the ability to do simple forms and menus. This collection capability is sufficiently limited to allow ease of implementation by different user-interface packages or windowing systems. However, the collection routines must support simple (up to eight item) menu or form styles of processing. On dumb terminals, forms processing becomes a set of one line questions. Without this capability, the application needs to be modified to support new security questions.

20.16 Packaging Products for the SIA

The SIA defines the security mechanism components that are required to port to the Tru64 UNIX system. These components are as follows:

- A shared library containing the mechanism-dependent (`siad_*()`) routines used as an interface to commands and utilities
- A default `/etc/sia/matrix.conf` file, which is installed to use the security mechanism through SIA

The shared library must contain all of the `siad_*()` routines described in Table 20–3. The default dummy routine for any `siad_*()` routine always returns the SIADFAIL failure response. If a security mechanism is supplying dummy routines, these routines should not be configured into the `matrix.conf` file.

The `/etc/sia/matrix.conf` file contains one line for each `siad_*()` routine. This line contains the mechanism identifiers (called `mech_types`) and the actual path to the security mechanism library. The `sia_*()` routines use this set of keys to call mechanisms in a right to left ordering. Example 13–1 illustrates the default `matrix.conf` settings for Tru64 UNIX.

If the DCE security mechanism is to be called first followed by the BASE (BSD) security mechanism, the configuration line for `siad_init()` might look like the following:

```
siad_init=(DCE,/usr/shlib/libdcesia.so)(BSD,libc.so)
```

Layered security products must deliver pretested matrix.conf files on their kits. The modification of an SIA matrix.conf file must be followed by a reboot. System administrators must never be required to edit a live matrix.conf file hand.

See Chapter 13 for a more detailed discussion of the matrix.conf file.

20.17 Security Mechanism-Dependent Interface

Security mechanisms are required to provide all of the siad_*() entry points. (See Table 20–3.) The default stub routine should return SIADFAIL. With the exception of the session routines, no stubs should ever be called in the /etc/sia/matrix.conf file. The session routines must all have the same number and order of mechanism to keep the mechanism index (mechind) consistent. However, if an error in configuration occurs, the stub routines deliver the appropriate SIADFAIL response.

The order of security mechanisms in the /etc/sia/matrix.conf file is the same for each class of interfaces. Therefore, if a security mechanism supports session processing, it is called in the same order for all the session related interfaces.

The layered security mechanism should provide a set of private entry points prefixed by *mechanism_name__* for each of the siad_*() entries used for internal calls within the mechanism to siad_*() routines. An example of this is in the BASE mechanism in libc. To assure that the BASE mechanism is calling its own siad_getpwuid() routine, a separate entry point is created and called from the siad_getpwuid() entry as follows:

```
int siad_getpwuid(uid_t uid, struct passwd *result, \
                                    char *buffer, int buflen)
        {
        return(bsd_siad_getpwuid(uid,result,buffer,buflen));
        }

static int bsd_siad_getpwuid(uid_t uid, struct passwd *result, \
                                         char *buffer, int buflen)
        {
        /* The BSD security mechanism siad_getpwuid() routine */
        }
```

If the convention of supplying internal names is used for all of the siad_*() entry points, a layered security mechanism can then produce a separate library containing all the security mechanism-dependent code. This leaves the configured shared library with only stubs that call the other library.

Security mechanisms generally fall into two categories: local and distributed. The local security mechanism is responsible for establishing all of the local context required to establish a session on the local system. There are two

local security mechanisms in Tru64 UNIX: the BASE mechanism and the ENHANCED mechanism.

Distributed mechanisms, like DCE, are more concerned with establishing distributed session context like Kerberos tickets. However, the distributed security mechanism may provide some local context that can be used by the local security mechanism. The distributed security mechanism may also provide a sufficiently strong authentication to allow a local mechanism to trust it for authentication. This notion of one mechanism trusting another is called vouching and allows the user to be authenticated only once to establish a login session. Local mechanisms should always be configured last in the calling sequences.

All of the SIA capabilities listed in this section can be configured to use multiple security mechanisms.

20.18 Single-User Mode

If you want to have your own single-user security mode, you need to rebuild and replace the commands and utilities affected, such as any statically linked binaries found in `/sbin`. This can be accomplished by providing an `siad_*()` routine library to precede `libc` in the link order for the affected commands.

The new routines need to override the `__siad_*()` routines, as opposed to the `siad_*()` routines. The `siad_*()` naming convention is the weak symbol entry point, while the `__siad_*()` convention is the strong symbol entry point that is actually used. See Appendix D for more information about routine-naming conventions.

21

Programming with ACLs

This chapter discusses the following topics:

- An introduction to access control lists (ACLs)
- ACL data representations
- ACL Library Routines
- Rules for creating, replicating, and validating ACLs
- ACL creation example
- ACL inheritance example

21.1 Introduction to ACLs

Tru64 UNIX access control lists (ACLs) are an optional extension to the discretionary access control (DAC) traditionaly provided on a UNIX system. Traditional UNIX DAC is the traditional UNIX permission bits; ACLs are an extension of the UNIX permission bits. A file or directory that has only the permission bits may be considered an object with an ACL containing only the three required or base entries that correspond to the `usr`, `group`, and `other` permission bits.

There are two types of ACLs:

- An **access ACL** is associated with a file or directory, and is used to determine if a process may access the file or directory.
- **Default ACLs** are associated with a directory. Default ACLs are used to determine the ACLs applied to new files and subdirectories created in the given directory. See Section 21.6 for more information.

The Tru64 UNIX ACL implementation is based on Draft 13 with some Draft 15 extensions of the POSIX P1003.6 standard.

ACLs can be applied to any file or directory on a file system that supports property lists. The file systems that support property lists are:

- UFS
- AdvFS
- NFS (between Tru64 UNIX systems)

ACLs can be applied even if ACL processing is not enabled on the system; however, ACL access checks and default ACL inheritance do not take place.

See Chapter 5 and Chapter 11 for a more detailed description of using and administering ACLs. See the `acl(4)` reference page for more information on using and programming with ACLs. See the `proplist(4)` reference page for more information on property lists.

21.2 ACL Data Representations

An ACL has an internal and an external representation. The external representation consists of text and is used to enter and display ACLs. Library routines manipulate ACLs in working storage in an internal representation that is only indirectly accessible to the calling routine. This internal representation can be interpreted with the `acl.h` header file.

21.2.1 Internal Data Representation

The ACL routines manipulate the working storage representation, which is a set of opaque data structures for ACLs and ACL entries. Your program should operate on these data structures only through the defined routines. Because the working storage data structures are subject to change, the interface is the only reliable way to access the data.

The working storage representation is not contiguous in memory. Also, a program cannot determine the sizes of ACL entries and ACL descriptors. The working storage data structures contain internal pointer references and are therefore meaningless if passed between processes or stored in a file. A program can convert the working storage representation of an ACL to other representations of an ACL.

The two types most commonly used to access the opaque data are `acl_t`, a pointer to type `acl` (the ACL structure), and `acl_entry_t`, a pointer to an ACL entry structure.

Note

The structures in the following sections are opaque internal data structures that are subject to change. Always use the defined types and the supplied library routines to access these structures.

The internal representation uses the following basic types and data structures.

21.2.1.1 typedef struct acl *acl_t;

The `acl_t` type is used to specify an internal (working storage) format ACL.

```
struct acl {
  int          acl_magic;      /* validation member */
  int          acl_num;        /* number of actual acl entries */
  int          acl_alloc_size; /* size available in the acl */
  acl_entry_t  acl_current;    /* pointer to current entry in */
  acl_entry_t  acl_first;      /* pointer to ACL linked list */
  attribute_t  *attr_data;     /* Pointer to the attr data */
};
```

21.2.1.2 typedef struct acl_entry *acl_entry_t;

The acl_entry_t type is used to specify an entry within an ACL.

```
struct acl_entry{
   acle_t            *entry;
   void              *head;
   struct acl_entry  *next;
   struct acl_entry  *prev;
   int               acl_magic;
   int               size;
};
```

21.2.1.3 typedef uint_t acl_type_t;

The ACL types supported are as follows:

```
#define  ACL_TYPE_ACC  0
#define  ACL_TYPE_ACCESS  ACL_TYPE_ACC
 /* The ACL is an access ACL.  The property list
    entry name for an access ACL is "DEC_ACL_ACC" */

#define  ACL_TYPE_DEF  1
#define ACL_TYPE_DEFAULT  ACL_TYPE_DEF
 /* The ACL is a default access ACL. The property list
    entry name for a default access ACL is "DEC_ACL_ACC" */
#define  ACL_TYPE_DEF_DIR  2
#define ACL_TYPE_DEFAULT_DIR  ACL_TYPE_DEF_DIR
 /* The ACL is a default directory ACL. The property list
    entry name for a default directory ACL is "DEC_ACL_DEF_DIR" */
```

acl_type_t is used to specify the ACL type.

21.2.1.4 typedef uint acl_tag_t;

The acl_tag_t type is used to specify the tag (the type) of an ACL entry. ACL entries with a tag type of ACL_USER or ACL_GROUP also have an associated tag qualifier. The tag qualifier is the ID of the user or group. The ACL entry tag types supported are:

```
#define  ACL_USER_OBJ  0
 /* entry that equates to the owning user permission bits. */
#define  ACL_GROUP_OBJ  1
 /* entry that equates to the owning group permission bits. */
#define  ACL_OTHER  2
#define  ACL_OTHER_OBJ  ACL_OTHER
 /* entry that equates to the other permission bits. */
#define  ACL_USER  23
 /* entry specifying permissions for a given user.  */
#define  ACL_GROUP  24
 /*  entry specifying permissions for a given group.  */
```

21.2.1.5 typedef uint_t acl_perm_t;

The `acl_perm_t` permission bit definitions are as follows:

```
#define  ACL_EXECUTE  0X001
#define  ACL_WRITE    0X002
#define  ACL_READ     0X004
```

21.2.1.6 typedef acl_perm_t *acl_permset_t;

The `acl_permset_t` type is used to point to the permissions assigned to an ACL entry.

21.2.1.7 Contiguous Internal Representation ACL

There is also a contiguous persistent data type for an ACL. This representation should be used only when the internal format ACL must persist between processes.

21.2.2 External Representation

The human-readable external representation of an ACL consists of a sequence of lines, each of which is terminated by a new-line character. The POSIX routines use the external representation when converting between the working storage representation and the text package.

The external representation is described in Section 5.6. Table 21–1 shows the structure of individual entries.

Table 21–1: ACL Entry External Representation

Entry Type	`acl_tag_t` Value	Entry
base user	`USER_OBJ`	user::*perms*
base group	`GROUP_OBJ`	group::*perms*
base other	`OTHER_OBJ`	other::*perms*
user	`USER`	user:*user_name*:*perms*
group	`GROUP`	group:*group_name*:*perms*

21.3 ACL Library Routines

The ACL routines are contained in the `libpacl.a` library. The ACL library routines are based on Draft 13 of the POSIX P1003.6 standard. See the reference page for each individual routine for detailed information.

The following routines are used to get, set, and validate ACLs:

`acl_valid()`	Checks the specified internal representation ACL for valid format.
`acl_delete_def_fd()`	Deletes the default access ACL from the designated directory using the file descriptor.
`acl_delete_def_file()`	Deletes the default access ACL from the designated directory.
`acl_get_fd()`	Retrieves the internal representation of the specified ACL type associated with the specific file or directory using the file descriptor.
`acl_get_file()`	Retrieves the internal representation of the specified ACL type associated with the specific file or directory.
`acl_set_fd()`	Sets the specified ACL type on the given file or directory to the specified ACL internal representation using the file descriptor.

`acl_set_file()`	Sets the specified ACL type on the given file or directory to the specified ACL internal representation.

The following routines retrieve and manipulate ACL entries:

`acl_copy_entry()`	Copies an ACL entry into the memory provided.
`acl_create_entry()`	Creates an empty ACL entry for the given ACL, allocating memory as necessary.
`acl_delete_entry()`	Deletes the designated ACL entry from an ACL.
`acl_first_entry()`	Resets the current ACL entry so that the next call to `acl_get_entry()` returns the first entry.
`acl_get_entry()`	Returns a pointer to the next ACL entry of the given ACL.

The following routines retrieve and manipulate fields in an ACL entry:

`acl_add_perm()`	Adds a permission to a set of permissions belonging to an ACL entry.
`acl_clear_perm()`	Clears a permission in a given ACL entry.
`acl_delete_perm()`	Removes permissions from a set of permissions belonging to an ACL entry.
`acl_get_permset()`	Copies the permissions from a given ACL entry to the location provided.
`acl_get_qualifier()`	Returns a pointer to the tag qualifier (ID) associated with a given ACL entry.
`acl_get_tag_type()`	Copies the tag (type) from the given ACL entry to the location provided.
`acl_set_permset()`	Sets the permissions in a given ACL entry to the given permissions.

`acl_set_qualifier()`	Sets the tag qualifier (ID) of the specified ACL entry to the given UID or GID.
`acl_set_tag_type()`	Sets the tag (type) of the specified ACL entry to the given type.

The following routines manage working storage for the ACL manipulation:

`acl_free()`	Releases all working storage associated with the given ACL.
`acl_free_qualifier()`	Releases working storage associated with the given tag qualifier.
`acl_free_text()`	Releases the buffer associated with the given external representation (text) ACL.
`acl_init()`	Allocates and initializes ACL internal representation working storage.
`acl_copy_ext()`	Copies the working storage internal format ACL data to the contiguous persistent ACL format.
`acl_copy_int()`	Copies contiguous persistent ACL data to working storage.
`acl_dup()`	Creates a copy of the designated ACL. The copy is independent of the original entry.
`acl_size()`	Calculates the size of the given ACL.

The following routines convert ACLs between external and internal representations:

`acl_from_text()`	Creates an internal representation ACL from the given external representation (text) ACL.
`acl_to_text()`	Creates an external representation (text) ACL from the given internal representation ACL.

21.4 ACL Rules

Some interactions between the ACL and the UNIX permissions are subtle. Unless you understand the interaction between ACL routines and the system calls that manipulate UNIX DAC attributes, you might get different permissions than you intended.

The following sections describe rules for programs that handle ACLs.

21.4.1 Object Creation

If ACLs are enabled and are supported on the file system, the `open()`, `creat()`, and `mkdir()` functions perform ACL inheritance when creating a file or directory. See Section 5.7 for a description of ACL inheritance.

When ACL inheritance is performed, the permissions on a created file come from the mode you provide and the inherited ACL, not the `umask`. Therefore, your program must set the mode when creating files and directories. The program must not depend on `umask` to protect the files and directories.

When copying one file to another, it is a common practice for a program to create a new file and propagate the owner, group, and mode. If the source file has an ACL, your program should propagate that ACL to the target file in all cases where the mode is propagated.

21.4.2 ACL Replication

Programs that replicate permissions must preserve the ACL. The discretionary protection of a file or directory is no longer described by the owner, group, and permissions; it includes the ACL which is a superset of the permissions. Neglecting to copy the ACL could allow unintended access to the file or directory.

21.4.3 ACL Validity

Any ACL you create must be valid according to the following POSIX ACL rules:

- It must have at least the three base entries
- The user entries must have unique valid qualifiers
- The group entries must have unique valid qualifiers
- The user and group identifiers must be valid

You can use the `acl_valid()` routine to check your ACLs.

21.5 ACL Creation Example

Assume that you want to set a file's access ACL to the following permissions:

```
user::rwx
user:june:r-x
user:sally:r-x
group::rwx
group:mktg:rwx
other::r-x
```

The following code takes a tabular form of the ACL, creates a working storage representation of the ACL, and applies it to a file. If you extract the following code into a file named `acl_example.c`, use the following command to compile it:

```
# cc -o acl_example -lpacl -lsecurity acl_example.c
```

```
#include <unistd.h>
#include <sys/types.h>
#include <prot.h>
#include <errno.h>
#include <sys/acl.h>

struct entries {
    acl_tag_t       tag_type;
    char            *qualifier;
    acl_perm_t      perms;
} table[] = {
    /* An ACL must (at a minimum) have the three base */
    /* entries that correspond to the permission bits */
    { ACL_USER_OBJ,  NULL,    ACL_READ | ACL_WRITE | ACL_EXECUTE },
    { ACL_USER,      "june",  ACL_READ | ACL_EXECUTE },
    { ACL_USER,      "sally", ACL_READ | ACL_EXECUTE },
    { ACL_GROUP_OBJ, NULL,    ACL_READ | ACL_WRITE | ACL_EXECUTE },
    { ACL_GROUP,     "mktg",  ACL_READ | ACL_WRITE | ACL_EXECUTE },
    { ACL_OTHER_OBJ, NULL,    ACL_READ | ACL_EXECUTE },
};

#define TABLE_ENTRIES (sizeof(table)/sizeof(table[0]))

main ( argc, argv )
int argc;
char *argv[];
{
    acl_t acl_p;
    acl_entry_t entry_p;
    acl_entry_t check_p;
    int     i, ret;
    uid_t   uid;
    gid_t   gid;
```

```
/* Did the user enter a filename? */
if (argc != 2) {
     printf("Usage: %s filename\n",argv[0]);
     exit(1);
}

/* Check to see if ACLs are supported and enabled for filename */
ret = pathconf(argv[1], _PC_ACL_EXTENDED);              /*1*/
if (ret == 1) {
     printf("  ACLs are enabled for file %s\n",argv[1]);
}
else if (ret == 0) {
     printf("  ACLs are supported for file %s,\n",argv[1]);
     printf("  but ACLs are not currently enabled on \n");
     printf("  the system\n");
}
else if ((ret == -1) && (errno == EINVAL)) {
     printf("  ACLs not supported on filesystem\n");
     exit(1);
}
else {
     printf("  Error checking file ACL status for \n");
     printf("  file %s, exiting...\n",argv[1]);
     perror("pathconf");
     exit(1);
}


/* Allocate an ACL */
acl_p = acl_init(1024);                    /* 2 */

/* Walk through the table creating corresponding ACL entries */
for(i=0;i<TABLE_ENTRIES;i++) {

/* Initialize the entry */
entry_p = acl_create_entry(&acl_p);            /* 3 */

  /* Set the permissions for the entry */
acl_set_permset(entry_p,&table[i].perms);        /* 4 */

/* Set the user or group information for the entry */
switch(table[i].tag_type) {

   case ACL_USER:

       /* Get the uid from the user name */
       uid=pw_nametoid(table[i].qualifier);        /* 5 */
       if (uid == (uid_t) -1) {
           printf("  No translation for user name %s\n",
           table[i].qualifier);
```

```
            printf("  Exiting...\n");
            exit(1);
        }

        /* Specify this is a "USER:" entry */
        acl_set_tag_type(entry_p,table[i].tag_type);    /*[6]*/

        /* Set the uid (entry qualifier) */
        acl_set_qualifier(entry_p,(void *)&uid);    /*[7]*/
        break;

    case ACL_GROUP:

        /* Get the gid from the group name */
        gid=gr_nametoid(table[i].qualifier);        /*[8]*/
        if (gid == (gid_t) -1) {
            printf("  No translation for group name %s\n",
            table[i].qualifier);
            printf("  Exiting...\n");
            exit(1);
        }

        /* Specify this is a "GROUP:" entry */
        acl_set_tag_type(entry_p,table[i].tag_type);

        /* Set the gid (entry qualifier) */
        acl_set_qualifier(entry_p,(void *) &gid);
        break;

    default:

        /* The three entries corresponding to the */
        /* Permission bits don't have qualifiers */
        acl_set_tag_type(entry_p,table[i].tag_type);
        acl_set_qualifier(entry_p,NULL);
        break;
    }
}

/* Is the created ACL valid? */
if (acl_valid(acl_p, &check_p) < 0) {           /*[9]*/
    printf("  Not Valid ACL\n");
    if (check_p) printf("  Duplicate entries\n");
    printf("  Exiting...\n");
    exit(1);
}

/* Set the ACL on the file */
if (acl_set_file(argv[1],ACL_TYPE_ACCESS, acl_p) < 0)
     perror("acl_set_file");
```

```
    /* Free the storage allocated for the ACL */
    acl_free(acl_p);
}
```

1 The _PC_ACL_ENABLED attribute of `pathconf()` returns the status of ACL processing for the given file.

2 This demonstrates the use of the initialization call for a working storage representation of the ACL. If the storage allocated is not big enough for all of the entries in the completed ACL, the `acl_create_entry()` routine will allocate more memory.

3 A new ACL entry is allocated with this call. The tag type, qualifier, and permissions in this new entry are unspecified.

4 The `acl_set_permset()` routine sets the permissions for the ACL entry.

5 The `pw_nametoid()` routine is an optimized mapping from user name to user ID and works with either Base or Enhanced security enabled. The `pw_nametoid()` routine is described in the `pw_mapping(3)` reference page.

6 The `acl_set_tag_type()` function sets the type for the given ACL entry. The current tag types are: ACL_USER_OBJ (owner permission bits), ACL_GROUP_OBJ (group permission bits), ACL_OTHER_OBJ (other permission bits), ACL_USER (permissions for specified user), ACL_GROUP (permissions for specified group).

7 The `acl_set_qualifier()` function sets the ID for the ACL_USER and ACL_GROUP tag types. This specifies which user or group the entry refers to. The other tag types do not require an ID.

8 The `gr_grouptoid()` routine provides an optimized mapping from group name to group ID and works with either Base or Enhanced security enabled. It is described in the `pw_mapping(3)` reference page.

9 The `acl_valid()` routine checks for missing and duplicate entries.

21.6 ACL Inheritance Example

This section shows how a program can specify a default access ACL on a directory and then describes what happens when a file and a directory are created in that directory. There is another type of default ACL called a default directory ACL. ACLs are inherited differently if a directory has a default directory ACL in addition to or in place of a default access ACL. See Section 5.7 for a complete description of the ACL inheritance rules.

Assume that directory `/usr/john/acl_dir` has the following access and default access ACLs:

```
% getacl /usr/john/acl_dir

# file: /usr/john/acl_dir
# owner: john
# group: prog
#
user::rwx
user:june:r-x
user:fred:r-x
group::rwx
group:mktg:rwx
other::r-x

% getacl -d /usr/john/acl_dir

# file: /usr/john/acl_dir
# owner: john
# group: prog
#
user::rwx
user:june:r-x
user:sally:r-x
group::rwx
group:mktg:rwx
other::rwx
```

The following program can be used to update the default access ACL on a directory to remove read and write permissions from a group entry and then create a regular file and a directory in the given directory. If you extract the following code into a file named `acl_inheritance.c`, it can be compiled with the following command:

```
% cc -o acl_inheritance -lpacl -lsecurity acl_inheritance.c
```

```
#include <unistd.h>
#include <sys/types.h>
#include <prot.h>
#include <errno.h>
#include <sys/acl.h>

#define REGULAR_FILE "regular"
#define DIRECTORY_FILE "dir"

main (argc, argv)
int argc;
char *argv[];
{
    acl_permset_t  acl_permset;
    gid_t          *qualifier = NULL;
    acl_tag_t      tag_type;
    acl_t          acl;
```

```
acl_entry_t    acl_entry;
gid_t          my_gid;
int            ret;
char           pathname[PATH_MAX + 1];
int            fd;

/* Did the user enter a directory name and group name? */
if (argc != 3) {
   printf("Usage: %s directory group\n",argv[0]);
   exit(1);
}

/* Map the group name to a gid */
my_gid = gr_nametoid(argv[2]);
if (my_gid == (gid_t) -1) {
    printf("No translation for group %s\n",argv[2]);
    exit(1);
}

/* Read the default ACL from the directory */
acl = acl_get_file(argv[1], ACL_TYPE_DEFAULT);
if (!acl) {
  if (errno) {
     perror("acl_get_file");
  }
  else {
     printf("No default ACL found on %s\n", argv[1]);
  }
  exit(1);
}

ret = acl_first_entry(acl);
if (ret) {
   perror("acl_first_entry");
   exit(1);
}

/* Scan the ACL looking for the entry */
while (acl_entry = acl_get_entry(acl)) {

   /* retrieve the entry type */
   ret = acl_get_tag_type(acl_entry, &tag_type);
   if (ret) {
      perror("acl_get_tag_type");
      exit(1);
   }

   if (tag_type != ACL_GROUP) continue;

   qualifier = (gid_t *)acl_get_qualifier(acl_entry);
```

```
   if (!qualifier) {
      perror("acl_get_qualifier");
      exit(1);
   }

   /* Check for appropriate entry */
   if (*qualifier != my_gid) continue;

   ret = acl_get_permset(acl_entry, &acl_permset);
   if (ret) {
      perror("acl_get_permset");
      exit(1);
   }

   *acl_permset = *acl_permset & ~(ACL_READ | ACL_WRITE);

   ret = acl_set_permset(acl_entry, acl_permset);
   if (ret) {
      perror("acl_set_permset");
      exit(1);
   }

   ret = acl_set_file(argv[1], ACL_TYPE_DEFAULT, acl);
   if (ret) {
      perror("acl_set_file");
      exit(1);
   }

   break;

}

if (!acl_entry) {
   if (errno) {
      perror("acl_get_entry");
   }
   else {
      printf("ACL entry for %s not found\n", argv[2]);
   }
   exit(1);
}

/* Create the regular file */
sprintf(pathname, "%s/%s", argv[1], REGULAR_FILE);

fd = creat(pathname, 0644);
if (fd == -1) {
   perror("creat");
   exit(1);
}
```

```
    close(fd);

    /* Create the directory */
    sprintf(pathname, "%s/%s", argv[1], DIRECTORY_FILE);

    ret = mkdir(pathname, 0700);
    if (ret == -1) {
       perror("mkdir");
       exit(1);
    }
}
```

When you run the previous example program, it removes the read and write permissions from the `mktg` group in the default ACL shown above. The program then creates a regular file and a directory in that directory to demonstrate ACL inheritance. Enter the following command to execute the example program:

```
% ./acl_inheritance /usr/john/acl_dir mktg
```

When the example program is executed, the access ACL on the newly created file and the access and default access ACLs on the newly created directory are as follows:

```
% getacl /usr/john/acl_dir/regular
# file: /usr/john/acl_dir/regular
# owner: john
# group: prog
#
user::rw-
user:june:r-x
user:sally:r-x
group::r--
group:mktg:--x
other::r--
```

Note that the permissions for the owning user, the owning group, and other are set to the logical AND of the default access ACL and the mode specified with the `creat()` call. The `umask` is not used when ACL inheritance takes place. The other entries are taken from the default access ACL of the parent directory, not the access ACL.

```
% getacl /usr/john/acl_dir/dir
# file: /usr/john/acl_dir/dir
# owner: john
# group: prog
#
user::rwx
user:june:r-x
user:sally:r-x
group::---
group:mktg:--x
other::---
```

Note that the access ACL inheritance rules for a subdirectory created in a directory that has a default access ACL are the same as those for a file. This is true only if there is not a default directory ACL on the parent directory in addition to the default access ACL.

The following command line displays the default access ACL:

```
% getacl -d /usr/john/acl_dir/dir
# file: /usr/john/acl_dir/dir
# owner: john
# group: prog
#
user::rwx
user:june:r-x
user:sally:r-x
group::rwx
group:mktg:--x
other::rwx
```

Note that the default ACL is inherited from the directory's parent.

A

TCB File Summary

Table A–1 contains a summary of all the files that are in the trusted computing base (TCB) on the trusted Tru64 UNIX system. Most of these files are installed on the base system, some of the files are created during the installation process, and some are databases created by a running system. Characteristics of those files are included in the Remarks column of the table.

Table A–1: Trusted Computing Base Files

File Name	Remarks
/.cshrc	Root account csh startup script
/.login	Root account csh startup script
/.logout	Root account csh logout script
/.profile	Root account startup script
/vmunix	OS execution image
/dev/[rz][0-3][a-z]	Block device disk partitions
/dev/console	System console device used in single-user mode
/dev/kmem	Kernel memory pseudodevice
/dev/mem	Kernel memory pseudodevice
/dev/null	Bit bucket pseudodevice
/dev/pts/*	Pseudo-ttys
/dev/rrz[0-3][a-z]	Character device disk partitions
/dev/tty	Current terminal pseudodevice
/dev/tty[0-f]	Terminal devices
/dev/tty*	Pseudo-ttys
/etc/auth/system/default	System defaults database
/etc/auth/system/devassign	Device assignment database
/etc/auth/system/files	File control database
/etc/auth/system/subsystems	Printable names for protected subsystems

Table A–1: Trusted Computing Base Files (cont.)

File Name	Remarks
/etc/auth/system/ttys.db	Terminal control database
/etc/fstab	Contains file systems to be mounted
/etc/group	Groups database
/etc/inittab	System initialization control file
/etc/passwd	Accounts database
/etc/policy/acl/pconfig	Discretionary policy setup file
/sbin/arp	Address resolution protocol (networking)
/sbin/chown	Change file owner
/sbin/clri	Clear on-disk inode
/sbin/date	Display/change time of day
/sbin/df	Display file system free space
/sbin/fsck	File system consistency checker
/sbin/fsdb	File system debugger
/sbin/halt	Bring system down
/sbin/hostid	Display/set system host ID
/sbin/hostname	Display/set host name
/sbin/ifconfig	Display/change network interface config (BSD networking)
/sbin/kill	Send software signal to process
/sbin/killall	Kill all active processes
/sbin/mknod	Create special files
/sbin/mount	Mount file systems or display mount table
/sbin/newfs	Format disk partition
/sbin/ping	Send ICMP alive request (BSD networking)
/sbin/ps	Display process status
/sbin/rc[0-3].d	System setup scripts
/sbin/reboot	Reboot the system
/sbin/route	Manage route tables (BSD networking)
/sbin/savecore	Dump memory image after crash

Table A–1: Trusted Computing Base Files (cont.)

File Name	Remarks
/sbin/sh	Shell
/sbin/sulogin	Single-user root login password verifier
/sbin/swapon	Add swap devices
/sbin/umount	Unmount mounted file systems
/tcb/bin/pacld	Discretionary policy daemon
	Maintain ACL synonym database
/tcb/bin/auditd	Audit daemon
/tcb/bin/authck	Security database consistency checker
/usr/tcb/bin/edauth	Authcap database editor
/usr/tcb/bin/convauth	Convert auth databases
/usr/tcb/bin/convuser	Convert user profile
/tcb/bin/setacl	Change a file's access control list
/tcb/bin/epa	Set process attributes
/tcb/bin/init	Initial process, change run levels
/tcb/bin/integrity	Security file attribute checker
/tcb/bin/mkaud	Audit reinitialization control files
/tcb/bin/reduce	Print audit report
/tcb/bin/su	Establish user identity program
/tcb/files/PACLDBASE	Discretionary policy tag/IR database
/tcb/files/audit	Compaction file directory on root file system
/tcb/files/audit/audit_parms	Default audit control file
/tcb/files/audit/audit_select	Audit selection criteria
/tcb/files/audit/reports	Directory for audit report storage
/tcb/files/auditrparms	Directory for reduction selection files
/tcb/files/auth.db	Protected password database for system accounts
/var/tcb/files/auth.db	Protected password database for user accounts
/tmp	Temporary directory
/users	Parent of users home directory

Table A–1: Trusted Computing Base Files (cont.)

File Name	Remarks
/usr/bin/at	Delayed job submission
/usr/bin/atq	List delayed job submissions
/usr/bin/atrm	Remove delayed job submissions
/usr/bin/cancel	Cancel a print request
/usr/bin/chgrp	Change file group
/usr/bin/cpio	Perform single-level import/export
/usr/bin/crontab	Periodic job table submission
/usr/bin/csh	Root account shell
/usr/bin/finger	Display account information
/usr/bin/from	Display mail headers
/usr/bin/ipcs	Display system V IPC object status
/usr/bin/login	Login program
/usr/bin/lp	Submit print request
/usr/bin/lpr	Submit print request
/usr/bin/lprm	Cancel print request
/usr/bin/lpstat	Display print subsystem status
/var/spool/mail/	Mail directory
/usr/bin/mesg	Disable/enable terminal messages
/usr/bin/mt	Manipulate tape device
/usr/bin/newgrp	Change process group assignment
/usr/bin/nice	Run process with different priority
/usr/bin/passwd	Password change program
/usr/bin/rcp	Network copy (BSD networking)
/usr/bin/rlogin	Network login (BSD networking)
/usr/bin/rsh	Remote shell (BSD networking)
/usr/bin/tar	Perform single-level import/export
/usr/bin/write	Open connection to another user/window
/usr/lbin/acct/accton	Enable system accounting
/usr/lbin/ex3.7preserve	Preserve an interrupted edit session
/usr/sbin/cron	Delayed/periodic job daemon

Table A–1: Trusted Computing Base Files (cont.)

File Name	Remarks
/usr/sbin/dcheck	Directory check utility
/usr/sbin/dumpfs	Display superblock
/usr/sbin/edquota	Edit quota controls
/usr/sbin/fastboot	Bring system down
/usr/sbin/fasthalt	Bring system down
/usr/sbin/icheck	Inode check utility
/usr/sbin/link	Perform `link()` system call
/usr/sbin/lpc	Line printer control program
/usr/sbin/lpd	Line printer daemon
/usr/sbin/mkpasswd	Create binary database from /etc/passwd
/usr/sbin/ncheck	Display file associated with inode number
/usr/sbin/netstat	Display network statistics
/usr/sbin/nfsstat	Display NFS statistics (NFS)
/usr/sbin/quot	Disk quota maintenance command
/usr/sbin/quotacheck	Disk quota maintenance command
/usr/sbin/quotaoff	Disk quota maintenance command
/usr/sbin/quotaon	Disk quota maintenance command
/usr/sbin/renice	Change priority of running command
/usr/sbin/repquota	Disk quota report
/usr/sbin/shutdown	System shutdown program
/usr/sbin/trpt	System reporting program
/usr/sbin/tunefs	Change values in super block
/usr/sbin/vipw	Manipulate passwords
/etc/passwd	BASE security password file
/usr/sbin/wall	Send message to all logged in users
/usr/share/lib/sechelp/	Help files for user interface programs
/usr/shlib/libsecurity.so	Security-relevant library routines
/var/adm/cron/	Administrative control files for `cron`
/var/adm/pacct	Accounting file

Table A–1: Trusted Computing Base Files (cont.)

File Name	Remarks
/var/adm/utmp	Hold user and accounting information (current)
/var/adm/wtmp	Hold user and accounting information (since boot)

Table A–2 lists files that are installed on the trusted system but not on a nontrusted system, and files that are modified on a trusted system. The files in this table are not considered part of the trusted computing base.

Table A–2: Files Not in Trusted Computing Base

File Name	Remarks
/usr/include/*.h	Many files modified/added
/usr/include/sys/*.h	Many files modified/added

B

Auditable Events and Aliases

This appendix contains the default auditable events (/etc/sec/audit_events) and the default audit event aliases (/etc/sec/event_aliases) as they as delivered on Tru64 UNIX.

B.1 Default Auditable Events File

The following is the default /etc/sec/audit_events file:

```
!  Audited system calls:
exit                            succeed  fail
fork                            succeed  fail
old open                        succeed  fail
close                           succeed
old creat                       succeed  fail
link                            succeed  fail
unlink                          succeed  fail
execv                           succeed  fail
chdir                           succeed  fail
fchdir                          succeed  fail
mknod                           succeed  fail
chmod                           succeed  fail
chown                           succeed  fail
getfsstat                       succeed  fail
mount                           succeed  fail
unmount                         succeed  fail
setuid                          succeed  fail
exec_with_loader                succeed  fail
ptrace                          succeed  fail
nrecvmsg                        succeed  fail
nsendmsg                        succeed  fail
nrecvfrom                       succeed  fail
naccept                         succeed  fail
access                          succeed  fail
kill                            succeed  fail
old stat                        succeed  fail
setpgid                         succeed  fail
old lstat                       succeed  fail
dup                             succeed  fail
pipe                            succeed  fail
open                            succeed  fail
setlogin                        succeed  fail
acct                            succeed  fail
```

```
classcntl                       succeed  fail
ioctl                           succeed  fail
reboot                          succeed  fail
revoke                          succeed  fail
symlink                         succeed  fail
readlink                        succeed  fail
execve                          succeed  fail
chroot                          succeed  fail
old fstat                       succeed  fail
vfork                           succeed  fail
stat                            succeed  fail
lstat                           succeed  fail
mmap                            succeed  fail
munmap                          succeed  fail
mprotect                        succeed  fail
old vhangup                     succeed  fail
kmodcall                        succeed  fail
setgroups                       succeed  fail
setpgrp                         succeed  fail
table                           succeed  fail
sethostname                     succeed  fail
dup2                            succeed  fail
fstat                           succeed  fail
fcntl                           succeed  fail
setpriority                     succeed  fail
socket                          succeed  fail
connect                         succeed  fail
accept                          succeed  fail
bind                            succeed  fail
setsockopt                      succeed  fail
recvmsg                         succeed  fail
sendmsg                         succeed  fail
settimeofday                    succeed  fail
fchown                          succeed  fail
fchmod                          succeed  fail
recvfrom                        succeed  fail
setreuid                        succeed  fail
setregid                        succeed  fail
rename                          succeed  fail
truncate                        succeed  fail
ftruncate                       succeed  fail
setgid                          succeed  fail
sendto                          succeed  fail
shutdown                        succeed  fail
socketpair                      succeed  fail
mkdir                           succeed  fail
rmdir                           succeed  fail
utimes                          succeed  fail
adjtime                         succeed  fail
sethostid                       succeed  fail
```

```
old killpg                      succeed  fail
setsid                          succeed  fail
pid_unblock                     succeed  fail
getdirentries                   succeed  fail
statfs                          succeed  fail
fstatfs                         succeed  fail
setdomainname                   succeed  fail
exportfs                        succeed  fail
getmnt                          succeed  fail
alternate setsid                succeed  fail
swapon                          succeed  fail
msgctl                          succeed  fail
msgget                          succeed  fail
msgrcv                          succeed  fail
msgsnd                          succeed  fail
semctl                          succeed  fail
semget                          succeed  fail
semop                           succeed  fail
lchown                          succeed  fail
shmat                           succeed  fail
shmctl                          succeed  fail
shmdt                           succeed  fail
shmget                          succeed  fail
utc_adjtime                     succeed  fail
security                        succeed  fail
kloadcall                       succeed  fail
priocntlset                     succeed  fail
sigsendset                      succeed  fail
msfs_syscall                    succeed  fail
sysinfo                         succeed  fail
uadmin                          succeed  fail
fuser                           succeed  fail
proplist_syscall                succeed  fail
ntp_adjtime                     succeed  fail
audcntl                         succeed  fail
setsysinfo                      succeed  fail
swapctl                         succeed  fail
memcntl                         succeed  fail
SystemV/unlink                  succeed  fail
SystemV/open                    succeed  fail
RT/memlk                        succeed  fail
RT/memunlk                      succeed  fail
RT/psx4_time_drift              succeed  fail
RT/rt_setprio                   succeed  fail

!  Audited trusted events:
audit_start                     succeed  fail
audit_stop                      succeed  fail
auditconfig                     succeed  fail
audit_suspend                   succeed  fail
```

```
audit_log_change                succeed  fail
audit_log_creat                 succeed  fail
audit_xmit_fail                 succeed  fail
audit_reboot                    succeed  fail
audit_log_overwrite             succeed  fail
audit_daemon_exit               succeed  fail
login                           succeed  fail
logout                          succeed  fail
auth_event                      succeed  fail
audgen8                         succeed  fail
net_tcp_stray_packet            succeed  fail
net_tcp_syn_timeout             succeed  fail
net_udp_stray_packet            succeed  fail
net_tcp_rejected_conn           succeed  fail

!  Audited mach traps:
lw_wire                         succeed  fail
lw_unwire                       succeed  fail
init_process                    succeed  fail
host_priv_self                  succeed  fail
semop_fast                      succeed  fail

!  Audited mach ipc events:
task_create                     succeed  fail
task_terminate                  succeed  fail
task_threads                    succeed  fail
thread_terminate                succeed  fail
vm_allocate                     succeed  fail
vm_deallocate                   succeed  fail
vm_protect                      succeed  fail
vm_inherit                      succeed  fail
vm_read                         succeed  fail
vm_write                        succeed  fail
vm_copy                         succeed  fail
vm_region                       succeed  fail
task_by_unix_pid                succeed  fail
bind_thread_to_cpu              succeed  fail
task_suspend                    succeed  fail
task_resume                     succeed  fail
task_get_special_port           succeed  fail
task_set_special_port           succeed  fail
thread_create                   succeed  fail
thread_suspend                  succeed  fail
thread_resume                   succeed  fail
thread_set_state                succeed  fail
thread_get_special_port         succeed  fail
thread_set_special_port         succeed  fail
port_allocate                   succeed  fail
port_deallocate                 succeed  fail
port_insert_send                succeed  fail
```

```
port_extract_send               succeed  fail
port_insert_receive             succeed  fail
port_extract_receive            succeed  fail
host_processors                 succeed  fail
processor_start                 succeed  fail
processor_exit                  succeed  fail
processor_set_default           succeed  fail
xxx_processor_set_default_priv  succeed  fail
processor_set_tasks             succeed  fail
processor_set_threads           succeed  fail
host_processor_set_priv         succeed  fail
host_processors_name            succeed  fail
host_processor_priv             succeed  fail
```

B.2 Sample Event Aliases File

The following is the sample `/etc/sec/event_aliases` file provided with the Tru64 UNIX system:

```
# This is a SAMPLE alias list.  Your alias list should be built to
# satisfy your site's requirements.

obj_creat:  "old open" "old creat" link mknod open symlink mkdir \
     SystemV/open

obj_delete: unlink truncate ftruncate SystemV/unlink rmdir

exec:       execv exec_with_loader execve

obj_access: access "old stat" "old lstat" "old open" open statfs \
     fstatfs readlink "old fstat" stat lstat fstat close:1:0 \
     dup dup2 fcntl "old creat" mmap munmap mprotect memcntl \
     SystemV/open

obj_modify: chmod chown fchown fchmod lchown utimes rename

ipc:        recvmsg nrecvmsg recvfrom nrecvfrom sendmsg nsendmsg \
     sendto accept naccept connect socket bind shutdown \
     socketpair pipe sysV_ipc kill "old killpg" setsockopt \
     sigsendset

sysV_ipc:   msgctl msgget msgrcv msgsnd shmat shmctl shmdt shmget \
     semctl semget semop

proc:       exit fork chdir fchdir setuid ptrace setpgid setlogin \
     chroot vfork setgroups setpgrp setpriority setreuid \
     setregid setgid audcntl RT/rt_setprio setsid "alternate \
     setsid" priocntlset

system:     getfsstat mount unmount acct reboot table sethostname \
     settimeofday adjtime sethostid setdomainname exportfs \
     getmnt swapon utc_adjtime audcntl setsysinfo kloadcall \
     getdirentries revoke "old vhangup" kmodcall security \
     sysinfo uadmin swapctl

misc:       ioctl msfs_syscall fuser

trusted_event: login logout auth_event audgen8
```

```
all:        obj_creat obj_delete exec obj_access obj_modify ipc \
     proc system misc trusted_event

#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

# adjtime is being called once a sec?

profile_audit:   audit_start:1:1 audit_stop:1:1 auditconfig:1:1 \
   audit_log_creat:1:1 audit_xmit_fail:1:1 \
                audit_reboot:1:1 audit_log_overwrite:1:1 \
   audit_daemon_exit:1:1 audcntl:1:1 settimeofday:1:1 \
   ntp_adjtime:1:1 utc_adjtime:1:1

profile_net:     connect:1:1 accept:1:1 bind:1:1

profile_auth:    login:1:1 logout:1:1 auth_event:1:1

profile_filesys: mount:1:1 unmount:1:1

profile_creat:   "old creat" link mknod symlink mkdir

profile_proc:    setuid setgid setlogin chroot setsid \
   "alternate setsid"

#================================================================
# Definition of catagories

# Desktop:
#  Provides suggested minimal auditing configuration for a single
#  user system.  Configuration provides monitoring of trusted audit
#  events, no monitoring of files, or network related events.
# ------------------------------------------------------------------
#  This alias assumes:
#   - Local access is primarily interactive login, generally limited
#     to one user at a time, activity tracked and controlled by the
#     system.
#   - Individual accountability is primarily maintained by the system.
#   - User related file area access is only limited by file owner
#     choice. Browsing is unrestricted.
#   - System related file areas are mostly readonly. Browsing is
#     unrestricted.
#   - Login uid is converted to username.
#   - Access to the network is monitored.
#   - Access to controlled files are unmonitored.
Desktop: \
  profile_audit \
  profile_auth

# Servers:
#  Provides suggested auditing configuration for a system which is
#  used as a server for networked based applications (such as
#  databases, web server, etc.). Configuration provides monitoring
#  of trusted # events, system files, network related files, and
#  network related events.
# ----------------------------------------------------------------
#  This alias assumes:
#   - Network access is restricted to application (mail, db server,
#     firewall, etc.) controlled access through network mechanisms
#     (TCP/IP reserved port, DECnet objects, etc.) with the
#     application being responsible for tracking activity.
#   - Interactive access is strictly controlled by the system, activity
#     is tracked by the system.
#   - Applications primarily handle access control, system control is
#     secondary.
```

```
#   - Local access logins are strictly controlled, activity is tracked
#     by the system.
#   - Individual accountability is primarily maintained by applications.
#   - User related file area access is strictly limited to application
#     related files. Browsing is controlled.
#   - System related file areas are at most read-only for user
#     aplication related functions. Browsing is controlled by
#     applications.
#   - Login uid is converted to username.
#   - Access to the network is monitored.
#   - Access to controlled files are monitored.
Server: \
  profile_audit \
  profile_auth \
  profile_net \
  profile_filesys \
  profile_proc \
  profile_creat obj_delete obj_modify

# Timesharing:
#  Provides suggested minimal auditing configuration for a system
#  which is used to support multiple interactive users. Configuration
#  provides monitoring of trusted events, no monitoring of system
#  files,or network related events or files.
# ------------------------------------------------------------------------
#  This alias assumes:
#   - Local access is primarily interactive login, activity is tracked
#     and controlled by the system.
#   - Individual accountability is primarily maintained by the system.
#   - Interactive logins are generally unrestricted.
#   - User related file area access is only limited by file owner
#     choice. Browsing is unrestricted.
#   - System related file areas are mostly readonly. Browsing is
#     unrestricted.
#   - Login uid is converted to username.
#   - Access to the network is unmonitored.
#   - Access to controlled files is unmonitored.
Timesharing: \
  profile_audit \
  profile_auth

# Timesharing_extended_audit:
#  Provides suggested auditing configuration for a system which is
#  used to support multiple interactive users. Configuration provides
#  monitoring of trusted events, system files, and no monitoring of
#  network related events or files.
# ------------------------------------------------------------------
#  This alias assumes:
#   - Local access is primarily interactive login, activity is tracked
#     and controlled by the system.
#   - Individual accountability is primarily maintained by the system.
#   - Interactive logins are generally unrestricted.
#   - User related file area access is limited only by file owner
#     choice. Browsing is unrestricted.
#   - System related file areas are mostly readonly. Browsing is
#     unrestricted.
#   - Access to the network is monitored.
#   - Access to controlled files is monitored.
Timesharing_extended_audit: \
  profile_audit \
  profile_auth \
  profile_filesys \
  profile_proc \
  profile_creat obj_delete obj_modify
```

```
# Networked_system:
#  Provides suggested auditing configuration for a system which
#  has networking enabled. Should be used in conjuction with
#  Desktop, Timesharing, or Timesharing_extended_audit templates.
#  Configuration provides monitoring of trusted events, network
#  related files and network related events.
# ------------------------------------------------------------------
#  This alias assumes:
#   - Network access is through application (mail, printer, etc.)
#     controlled network mechanisms (TCP/IP reserved port, DECnet
#     objects, etc.) which are responsible tracking activity and
#     controlling access, and Interative login with the system
#     tracking activity and controlling access.
#   - Access to the network is monitored.
#   - Access to controlled files is monitored.
Networked_system: \
  profile_audit \
  profile_net \
  profile_creat obj_delete obj_modify


# NIS_server:
#  Provides suggested auditing configuration for a system used as
#  a NIS server.  Should be used in conjuction with Desktop,
#  Timesharing, or Timesharing_extended_audit templates.
#  Configuration provides monitoring of trusted events, NIS
#  related files and network related events.
# ------------------------------------------------------------------
#  This alias assumes:
#   - Network access is through application (mail, printer, etc.)
#     controlled network mechanisms (TCP/IP reserved port, DECnet
#     objects, etc.) which are responsible tracking activity and
#     controlling access, and Interative login with the system
#     tracking activity and controlling access.
#   - NIS is enabled.
#   - Access to the network is monitored.
#   - Access to controlled files is monitored.
NIS_server: \
  profile_audit \
  profile_net \
  profile_creat obj_delete obj_modify
```

C

Coding Examples

The examples in this appendix illustrate how to use some of the routines in the trusted Tru64 UNIX system.

C.1 Source Code for a Reauthentication Program (sia-reauth.c)

Example C–1 is a program that performs password checking.

Example C–1: Reauthentication Program

```
#include <sia.h>
#include <siad.h>

#ifndef NOUID
#define NOUID ((uid_t) -1)
#endif

main (argc, argv)
int argc;
char **argv;
{
        int i;
 SIAENTITY *entity = NULL;
 int (*sia_collect)() = sia_collect_trm;
 char uname[32];
 struct passwd *pw;
 uid_t myuid;

 myuid = getluid();
 if (myuid == NOUID)
  myuid = getuid(); /* get ruid */
 pw = getpwuid(myuid);
 if (!pw || !pw->pw_name || !*pw->pw_name) {
  sleep(3); /* slow down attacks */
  (void) fprintf(stderr, "sorry");
  return 1;
 }
 (void) strcpy(uname, pw->pw_name);
 i = sia_ses_init(&entity, argc, argv, NULL, uname, \
                                              NULL, TRUE, NULL);
 if (i != SIASUCCESS) {
  sleep(3); /* slow down attacks */
  (void) fprintf(stderr, "sorry");
  return 1;
 }
 i = sia_ses_reauthent(sia_collect, entity);
 if (i != SIASUCCESS) {
  (void) sia_ses_release(&entity);
  sleep(3); /* slow down attacks */
  (void) fprintf(stderr, "sorry");
```

Example C–1: Reauthentication Program (cont.)

```
 return 1;
}
i = sia_ses_release(&entity);
if (i != SIASUCCESS) {
 sleep(3); /* slow down attacks */
 (void) fprintf(stderr, "sorry");
 return 1;
}

(void) fprintf(stderr, "Ok");

return 0;
}
```

C.2 Source Code for a Superuser Authentication Program (sia-suauth.c)

Example C–2 is a program that allows root to become a user to run daemons (such as `crontab` or `sendmail`) for the user.

Example C–2: Superuser Authentication Program

```
#include <sia.h>
#include <siad.h>

main (argc, argv)
int argc;
char **argv;
{
        int i;

        i = sia_auth(getuid());
        printf("result is %d", i);

}

int  sia_auth(uid)
int uid;
{

        char uname[32];
        static SIAENTITY *entity=NULL;
        static int oargc = 1;
        static char *oargv[1] = { "siatest" };
        static int (*sia_collect)()=sia_collect_trm;
 struct passwd *pw;

 pw = getpwuid(uid);
 if (!pw) {
     printf("getpwuid failure");
     return 8;
 }
 (void) strcpy(uname, pw->pw_name);
        printf("SIA authentication for uid: %d, uname: %s ", \
                                                        uid, uname);
        if (sia_ses_init(&entity,oargc,oargv,NULL,uname,NULL, \
```

Example C–2: Superuser Authentication Program (cont.)

```
                                        FALSE, NULL) == SIASUCCESS) {
                printf( "sia_ses_init successful");
                entity->authtype = SIA_A_SUAUTH;
  if (sia_make_entity_pwd(pw, entity) == SIASUCCESS) {
      printf("sia_make_entity_pwd successful");
  }
  else {
      printf("sia_make_entity_pwd un-successful");
  }
                if ((sia_ses_launch(NULL, entity)) == SIASUCCESS) {
                        printf( "sia_ses_launch successful");
                }
                else {
                        printf( "sia_ses_launch un-successful");
   entity = NULL;
                }
                if ((sia_ses_release(&entity)) == SIASUCCESS) {
                        printf( "sia_ses_release successful");
                }
                else {
                        printf( "sia_ses_release un-successful");
                        return(4);
                }

        }
        else {
                printf( "sia_ses_init un-successful");
                return(5);
        }
        printf( "sia  **** successful");
        return(6);
}
```

D

Symbol Preemption for SIA Routines

This appendix describes the naming convention for routines (added by developers) that must be followed to stay in compliance with ANSI C routine naming rules.

D.1 Overview of the Symbol Preemption Problem

Overriding the symbols used by the SIA routines in `libc` is not as simple as providing routines named the same as the SIA routines (such as, `siad_ses_init()`) in a library loaded before `libc.a`. This is because of the ANSI C convention for `libc` routine names and the symbols that must be reserved to the user.

A conflict exists between the requirements of ANSI C and the expectations of the application developers regarding what entry points can exist in the `libc.a` and `libc.so` libraries. The ANSI C standard lists the symbols allowed, and the only other symbols that are valid must be of the "reserved-to-vendor" form. That is, they must start with two underscores, or one underscore and a capital letter. This set of symbols is limited, and does not meet the expectations of the general user community.

D.2 The Tru64 UNIX Solution

To satisfy both ANSI C and developer expectations, Tru64 UNIX uses "strong" and "weak" symbols to provide the additional names. If a routine such as `bcopy()` is not allowed by ANSI C, it has a weak symbol named `bcopy()` and a strong symbol named `__bcopy()`.

The weak symbol can be preempted by the user with no effect on the `bcopy()` routine within `libc`, because the library uses the strong symbols for these "namespace-protected" routines.

For the SIA routines, this means that there is a weak symbol for `siad_ses_init` which is normally bound to the strong symbol `__siad_ses_init()`. If other code already uses the symbol `siad_ses_init()`, only the binding of the weak symbol is affected.

The SIA code in `libc` references the strong symbol `__siad_ses_init()` for its own uses. Thus, to override the default BASE security mechanism

for single-user mode, it is necessary to provide a replacement for the `__siad_ses_init()` routine.

For a library that is only dynamically loaded under the control of the SIA routines and the `/etc/sia/matrix.conf` file, it is only necessary to provide the `siad_ses_init()` form of the symbol name. If the dynamically loaded library is only used through the `matrix.conf` file, it is acceptable to provide both forms of symbols. This simplifies the code, but is not safe if the library usage ever changes to require that the library be linked against, not just dynamically loaded.

D.3 Replacing the Single-User Environment

Example D–1 shows the code to use if a security mechanism library developer needs to replace the single-user environment as well as provide a normal shared library for `matrix.conf`.

Example D–1: Preempting Symbols in Single-User Mode

```
/* preempt libc.a symbols in single-user mode */
#ifdef SINGLE_USER
# pragma weak siad_ses_init = __siad_ses_init
# define siad_ses_init __siad_ses_init
#endif
#include <sia.h>
#include <siad.h>
```

The single-user (static) library modules are then compiled as follows:

```
% cc -DSINGLE_USER ...
```

This keeps the shared library from interfering with the `libc.so` symbols, but allows the preemption of the `libc.a` symbols for the nonshared images used in single-user mode. The nonshared images are then built with the replacement mechanism library supplied to the linker before `libc.a` as in the following example:

```
% cc -non_shared -o passwd passwd.o -ldemo_mech
```

The shared library is built in the normal fashion.

E

C2 Level Security Configuration

This appendix provides a procedure for configuring your system to meet or exceed a C2 level of security as described in the *Orange Book*. When the system is used in accordance with a site security policy, a C2 network, and the appropriate physical security, a C2 level environment can be achieved.

You can configure your system to meet the minimum C2 requirements by following the instructions in Section E.3 or you can configure for the maximum practical level of security by using this entire document.

This appendix contains information on the following subjects:

- Evaluation status for Tru64 UNIX
- Site security policy
- Minimum C2 configuration
- Procedures to establish a secure installation of a Tru64 UNIX system
- Physical security
- Applications
- Periodic security administration procedures
- Reference documents and verification tools

E.1 Evaluation Status

The Tru64 UNIX operating system is delivered with an optional enhanced security subset. When this subset is installed and configured, the system is referred to as a trusted system. The Tru64 UNIX enhanced security features are designed to meet the C2 class of trust, as defined by the *Trusted Computer System Evaluation Criteria* (also called the *Orange Book*) An on-line version of the *Orange Book* is available at `http://nsi.org/Library/Compsec/orangebo.txt`.

The system is also designed to meet the F-C2 functional class, as defined in the *Information Technology Security Evaluation Criteria* (ITSEC).

The system's security mechanisms maintain full compatibility with existing Tru64 UNIX security mechanisms, while expanding the protection of user and system information.

Contact your sales representative for the latest evaluation and certification status of the Tru64 UNIX product.

E.2 Establishing a Security Policy

A security policy is a statement of the rules and practices that regulate how an organization maintains its computing environment and how the organization manages, protects, and distributes sensitive information.

An organization carries out its security policy by configuring the system as described in this procedure and by adhering to the administrative and procedural guidelines defined in the site policy.

Compaq recommends that you establish a written security policy for your site, as described in the *Site Security Handbook (RFC 1244)*.

Security consulting services are available from Compaq by calling (in the USA) 1.800.AT.COMPAQ. For more information, see the following Web site:

`http://www.compaq.com/services/internet/security/index.html`

To create your site's security policy do the following:

- Document the process for maintaining and changing the security policy.
- Establish the action taken by system administrators in the case of a break-in or other breach of security.
- Determine your site's audit policy including the following:
  - User activities you want to audit.
  - Locally defined audit events.
  - Location for audit logs.
  - Procedures for the review of audit logs.
  - Whether the auditing of login attempts to unrecognized account names (`login_uname`) is needed. Using this attribute can put passwords, entered out of sequence, with respect to the prompts, in the audit logs.
- Establish a service access policy (supervision, passwords).
- Determine the `umask` for your system (022 is the system default).
- Establish a schedule for verifying the integrity, including passwords, of your system and site.
- Define the boundaries of the system and the interfaces (`telnet`, `ftp`, for example) between the boundaries.
- Establish a magnetic media policy, especially for removable media.

- Determine what application software will be installed on your system and what its security implications are.
- Determine the password policy for your users. Some considerations follow:
  - Long passwords are hard to break, but inevitably they are written down.
  - If user-chosen passwords are used, only one person knows the password.
  - Machine-generated passwords are harder to break, but also harder to remember and will probably be written down.
- Determine the login controls for your system.
- Establish a procedure for system startups, shutdowns and upgrades.
- Establish a backup and recovery procedure.
- Determine who the system administrators (root access) are and exactly what their functions are to be.
- Establish one or more secure account prototypes (Local Templates) for creating user accounts using the Account Manager program.
- Establish a secure account template with startup files in the `/usr/skel` directory.
- Determine the access restrictions for each object on your system.
- Establish the groups for your system.
- Determine the access restrictions for each user on your system.
- Record for reference all the programs on your system that can set the UID or GID.
- Determine the export restrictions for file systems on your system.
- Establish change control procedures for subjects and objects on your system.
- Establish a procedure for physical access changes.
- Establish the physical access requirements for your system and console.
- Establish the physical access requirements for your network components.
- Establish your network security policy.
- Determine which remote access programs (`ftp`, `telnet`, and such) will run on your system.
- Determine the remote systems that will have access to your system. (Compaq recommends that you do not allow `.rhosts` and `.hosts.equiv` files.)

- Determine the console password requirements for your system and site.
- Establish a modem policy. (Consider authentication, the configuration for dial-in and dial-out access.)
- Create a "User Security Training" course or document for your site.
- Document how users will access the system: operating system, database, or application menu.
- Determine the secure devices for your site.

After your system is configured, the configuration files should change little and always in predictable ways. During periodic security reviews of your system, compare the base configuration files for content and permissions to the current files. Document the base system and network configuration by obtaining a listing of the following files and attaching them to the security policy:

```
/usr/skel/.profile
/usr/skel/.cshrc
/usr/skel/.login
/var/yp/<domain>/auto.master
/var/yp/<domain>/auto.home
/var/yp/<domain>/auto.###
/etc/auto.*
/etc/auth/*
/etc/dumpdates
/etc/ethers
/etc/exports
/etc/fstab
/etc/ftpusers
/etc/group
/etc/hosts
/etc/hosts.equiv
/etc/inetd.conf
/etc/motd
/etc/netgroups
/etc/passwd
/etc/profile
/etc/csh.login
/etc/logout          if used
/etc/remote
/etc/resolv.conf
/etc/rc.config
/etc/rc.site         optional, used with /etc/rc.config
/etc/screend.config
/etc/services
/etc/sec/site_events
/etc/sec/audit_events
/etc/sec/auditd_clients
/etc/sec/event_aliases
```

```
/etc/sec/auditd_cons
/etc/sec/audit_loc
/etc/securettys
/etc/svc.conf
/tcb/*
/usr/adm/messages
/var/spool/uucp/Permissions       if UUCP is active
/var/spool/uucp/Systems           if UUCP is active
/var/spool/uucp/remote.unknown    if UUCP is active
/var/adm/cron/at.allow
/var/adm/cron/at.deny
/var/adm/cron/cron.allow
/var/adm/cron/cron.deny
/var/adm/crontab/           any files in these directories
/var/tcb/*
/var/yp/src/*
```

E.3 Minimum C2 Configuration

Compaq's interpretation of the *Orange Book*'s requirements for a minimum C2 system is that the configuration for Tru64 UNIX is as follows:

- The requirement for a site security policy is met when you establish a security policy for your site as described in Section E.2 and the *Site Security Handbook (RFC 1244)*. Your security policy should be in written form.
- Users should be able to change their own passwords and the passwords should be machine generated. (Recommended by the *Green Book*.) See Section E.4.2 for password configuration details.
- The requirement for users to be notified of their last login is met when enhanced security is configured.
- The discretionary access control requirement is met by configuring ACLs (access control lists) on your system. See Section E.4.6 for configuration details. Compaq does not recommend using the `/usr/groups` approach for small systems (less than 32 users).
- The object reuse requirement mandates that workstations be configured with no `xhost` entries.
- Shared memory separation must be enabled. You do this by answering yes when `secconfig` asks if you want to disable segment sharing.
- The audit subsystem needs to be configured and available for use. Compaq recommends that, at a minimum, you run audit as described in Section E.4.5.

- The ability to verify the integrity of the trusted computing base (TCB) is met by running the `fverify` and `authck` commands periodically as determined by your site's security policy.

E.4 Initial Configuration

After you have installed the Tru64 UNIX software subsets (including the optional enhanced security and documentation extension subsets) onto your system, you will start the software configuration. During the configuration, several of the selections you make will affect the security of your system. The assumption is that you need the maximum practical security configuration for your system. The following sections document the areas of concern for security and Compaq's recommended configuration.

E.4.1 General Configuration

Compaq recommends the following general system configurations:

- Ensure that the `/tmp`, `/var/tmp`, and `/var/spool` directories are on a file system other than that of the root (`/`) and `/usr` directories.
- Do not run Netscape Navigator with JAVA enabled. Only enable JAVA when you are connected to known secure sites.
- Avoid connecting systems to the Internet whenever possible.

E.4.2 Enhanced Passwords and Authentication Using secconfig

Select the enhanced password attributes to match your site's security policy. See Section E.2 and Section 7.5.4 for details.

Compaq recommends the following password attributes (defaults are defined in the `/etc/auth/system/default` file):

- Select either user-chosen or machine-generated passwords and configure as follows:
  - For user-chosen passwords (`u_pickpw` field in the `/etc/auth/system/default` file), set the minimum length to 8 characters (`u_minlen#8`) and the maximum length to 80 characters (`u_maxlen#80`).
  - For machine-generated passwords (no `u_pickpw` field in the `/etc/auth/system/default` file), set the minimum length to 0 characters (`u_minlen#0`) and the maximum length to 10 characters (`u_maxlen#10`). The value of 0 for minimum length causes Tru64 UNIX to use the *Green Book* algorithm to generate passwords.
- Ensure that null passwords cannot be used (`u_nullpw@`)
- Set the password expiration time to 180 days (`u_exp#15724800`)

- Set the account lifetime set to 360 days (`u_life#31449600`)
- Set the depth of the password history file to 9 (`u_pwdepth#9`)
- Set the number of tries to enter a password before locking the account to 5 (`u_maxtries#5`)
- Set new accounts to be locked (`u_lock`)
- Set the maximum number of login attempts before the terminal is locked to 10 (`t_maxtries#10`)
- Set the delay between attempted logins to 2 seconds (`t_logdelay#2`)
- Select triviality checks (`u_restrict`) and site password restrictions (`u_policy`)

Use the Account Manager (`dxaccounts`) or the `edauth` program to change the default settings.

E.4.3 Libraries

The libraries on your system can be used in an attack. Secure the libraries as follows:

- Disable segment sharing by answering `yes` when prompted by `secconfig`.
- Verify that the permissions are correct (no write access except for the owner) and that the ownership is root on shared libraries (`/usr/shlib/*.so`), including any linked target files. Use the `ls -lL` command for this procedure.

E.4.4 Account Prototypes and Templates

The account templates used to create user account startup files are `/usr/skel/.login`, `/usr/skel/.cshrc` and `/usr/skel/.profile`.

Account prototypes (referred to as Local Templates) are provided by the Account Manager (`dxaccounts`). The prototypes let you set attributes like password expiration and login attempts for individual user accounts. If an attribute value is not specified in the local template, the value from the `default` file is used. The system-wide default attribute values are stored in the `/etc/auth/system/default` file. System default values are set with the `/usr/tcb/bin/edauth` command.

Configure user accounts as follows:

- Using the provided default templates, create account templates that reflect your site's security policy.
- Set the `umask` in the `/usr/skel/.login` file. (Compaq recommends a value of 027.)

- Designate a restricted shell (`Rsh`) for users where appropriate.
- Verify that each user has a valid entry path (login shell) on the system. Users can be placed directly into an application by executing the application from the user's `/home/.profile` or from the entry in the `/etc/passwd` file or as a start point for the user with the execution of a startup program.
- If user access is restricted through menu scripts called from the user's `.profile` file, the scripts should have a `trap` command at the beginning of the file to ensure that Ctrl/C and other keyboard interrupts are ignored.

E.4.5 Configuring the Audit Subsystem

Before the audit subsystem kernel option can be configured, it needs to be included for the kernel build. Use the `sysman auditconfig` utility to configure the audit subsystem any time after the kernel build. Compaq recommends that you configure and run audit as follows:

- Use the default location for audit logs (`/var/audit/auditlog.nnn`). For overflow protection, put the audit logs on a file system other than root (`/`) and `/usr`.
- Establish an alternate location for audit logs to provide for an overflow of audit log data by editing the `/etc/sec/auditd_loc` file.
- Send `auditd` messages to the console (`/dev/console`).
- Set the audit mask to audit `trusted_events` and to log the name of a user (as described in your site policy) who attempts to log in to an invalid account.

If you are starting the audit daemon from the command line, use the following command:

```
# /sbin/init.d/audit start
```

See Chapter 10 for audit configuration details.

E.4.6 Configuring ACLs

ACL processing can be dynamically enabled using the `sysconfig` command and can also be configured to be enabled automatically as part of system startup using the `sysconfigdb` command.

See the `sysconfig`(8) reference page and Chapter 11 for ACL configuration details.

E.4.7 Verifying That Your Installation Is Secure

After you have rebooted the system to enable the enhanced security options, run the `fverify` and `authck` programs to verify the integrity of your system.

E.4.8 Configuring Network Security

Proper network configuration is a critical part of your secure computing environment. Use the following checklist as an aid to network configuration:

- Do not use NIS (Network Information Services, formerly called Yellow Pages) to distribute root account information. See Section 9.5 for details.
- When using NIS, use the `/etc/yp/securenets` file, as described in the `ypserv`(8) reference page.
- Run `ypbind` with the `–S` flag and without the `–ypset` or `–ypsetme` options (default).
- If `uucp` is configured on your system, do the following:
  - Ensure that the `uucp` account is password controlled and that a separate `uucp` account is established for each machine that requires access.
  - Ensure that the `/var/spool/uucp/Permission` file has only valid entries.
  - Ensure that the `/var/spool/uucp/Systems` file has only valid entries.
- Ensure that the File Transfer Protocol (FTP) is secured and that, if possible, there are no anonymous FTP accounts. If you must use anonymous FTP, ensure the following:
  - The FTP account has an asterisk in the protected password field.
  - A `/usr/ftp` home directory is created for FTP. Create `/bin` and `/etc` subdirectories under the `/usr/ftp` directory.
  - Nothing in the home directory is owned by `ftp`.
  - A public subdirectory is created under the `/usr/ftp` directory for placement and retrieval of transferred files. User `ftp` should only have write access to the public subdirectory.
  - Create an `~ftp/etc/passwd` file with only the `ftp` account and no password.
  - Copy the `/etc/sia/bsd_matrix.conf` file to `~ftp/etc/sia/matrix.conf`.
  - Copy `/sbin/ls` to `~ftp/bin/ls`.
  - The login shell for the `ftp` account should be `/sbin/sink`.

- Ensure that workstations are using DES-cookie based authentication (default). See the XDM-AUTHORIZATION-1 parts of the dtlogin(1) reference page for more information.
- When /usr/bin/X11/xhost is run, nothing should be reported. The output should look like the following:

```
# xhost
access control enabled, only authorized clients can connect
#
```

E.4.9 Postinstallation Security Configuration

After the system is installed and configured, perform the activities in the following sections.

E.4.9.1 umask for Remote Access

Add a umask entry as described in your site security policy to the /etc/csh.login, /etc/profile, and /etc/init.d/inet files. (Note that the /etc/init.d/inet file is overwritten during an update installation.)

E.4.9.2 Devices

Using /usr/tcb/bin/dxdevices, create the devices with the security attributes that reflect your site's security policy.

Ensure that terminal ports are readable only by the owner by modifying the remote login shell file as follows:

Add the following to the /etc/profile file:

```
case "$TERM" in
none) ;;
*) /usr/bin/setacl –b `/usr/bin/tty` ;;
esac
```

Add the following to the /etc/csh.login file:

```
if ($?TERM) then
    if ("$TERM" != "none") then
 /usr/bin/setacl –b `/usr/bin/tty`
    endif
endif
```

See Chapter 8 for details dealing with devices.

E.4.9.3 Accounts

Compaq recommends that you create and verify accounts as follows:

- Create the user accounts for your system using either the Account Manager (/usr/bin/X11/dxaccounts) or by restoring the /usr/users area and associated files from a previous system.
- Ensure that home directories are mounted with the noexec, nosuid, and nodev options.
- Ensure that CDE users have the autopause feature enabled by using a command similar to the following:

  ```
  # grep extension.lockTimeout \
        ~/.dt/sessions/current/dt.resources
  ```

  A 0 status indicates that the autopause feature is disabled.
- Review the /etc/passwd and /var/tcb/files/auth.db databases to verify that user home directories and passwords are appropriate.

  See Chapter 9 for details on account creation.

E.4.9.4 Root Access

Because root access must be carefully controlled and monitored, make sure the following conditions are met:

- That all passwords are changed after a system installation or after support vendors have had access to your machine.
- That the root password is changed before vendor access is granted to prevent exposure of your password generation methodology.
- That the single-user password feature is enabled. See the sulogin(8) reference page.
- That using the su command to become root is logged by audit.
- That access to the setuid 0 or setgid 0 programs on your system is restricted (700, 710, or 711)
- That the /var/spool/cron/crontabs files are accessible only by root or the owner.
- That root access is restricted to certain devices for login or that users must use the su command to access the root account. See the securettys(4) reference page for more information.
- The logins for the system-supplied UIDs are limited (setting the u_lock field) where appropriate. The following table provides the restrictions recommended by Compaq:

UID	Recommended login Status
root	Restricted
daemon	Not allowed

UID	Recommended login Status
bin	Not allowed
sys	Not allowed
uucp	Restricted
nobody	Not allowed
adm	Restricted
lp	Not allowed

E.4.10 Network Configuration

Review the /etc/svc.conf file and ensure that a logical configuration has been set up for NIS. Also, if NIS is being used, verify that the client machines and the server have the correct domain name defined in the NIS_DOMAIN variable in the /etc/rc.config or /etc/rc.site file.

Ensure that the network files in the following table are protected:

File	Comment
/etc/exports	Validate the entries. Avoid using the –root= option if possible. Use the –access=<hostname> and –ro options on all specified file systems
/etc/hosts	
/etc/services	
/etc/protocols	
/etc/inetd.conf	
/etc/hosts.equiv	Validate that the entries are local hosts.
/etc/ethers	
~username/.rhosts	Remove these files or run rlogind and rshd with the –l flag set.

E.5 Physical Security

An important part of your site's security is the physical security of all the components in the environment. Check your physical security as follows:

- Verify that the system and its cabling are in a secure environment.
- Verify that all network components are physically secured. These include file servers, bridges, routers, hubs/concentrators, gateways, terminal servers, and modems.

- From the console prompt, ensure that the boot flags are set according to your site policy using the following command:

  ```
  >>> show
  ```

- If your system supports the console password feature, ensure that it is being used. Consult your hardware documentation for information on console password support.
- Verify that a console terminal's function keys have not been programmed for login or password information.
- Ensure that modems have an automatic disconnect feature. Also make sure modems are in a secure environment.

E.6 Applications

To ensure the security of application software running on your system, make sure that the following conditions are met:

- Restrict any `setuid` or `setgid` programs.
- Ensure that any control files and executable files are writable only by the root account.
- If a firewall product is installed, see the firewall's documentation for the appropriate configuration information.
- If you are running the `screend` program, configure as described in the `screend`(8) reference page.
- If you have tunneling software installed, ensure that it is secure, as described in its documentation.

E.7 Periodic Security Administration Procedures

The frequency of the different classes of review activities is determined by your site's security policy. Perform the following activities on a regular schedule:

- Back up the system and its applications.
- Review the audit logs.
- Review the system accounting logs.
- Run the `fverify` and `authck` programs to verify the integrity of your system. Some public domain programs, such as `cops` and `tripwire`, are useful to help verify system integrity.
- Verify that your system has only necessary and authorized programs.
- Verify that compilers are available only on systems used for development.

- Verify that your system has only authorized, root-owned `setuid` and `setgid` programs using the following command:

  ```
  # find / \( -perm -4000 -o -perm -2000 \) -ls
  ```

- Review the `/etc/exports` file to verify that all entries are valid.
- Check your user accounts as follows:
  - Review the `/etc/passwd` file to verify that all accounts are still valid.
  - Run the following command to ensure there are no enhanced profiles (`prpasswd` entries) without `/etc/passwd` entries:

    ```
    # /usr/tcb/bin/convuser -dN
    ```

  - Verify that the home directory permissions are set according to your site policy.
  - Verify that all files in a user's home directory are owned by that user.
  - Verify that each user has a valid entry path (login shell) on the system.
  - Verify that entries in a `.rhosts` or `.netrc` file are appropriate. Verify that that any `.exrc` and `.netrc` files are located only in user home directories.
  - Review the `hosts.equiv` file for valid entries.
  - Ensure that entries in the following files do not conflict with system parameters and that the files are protected by a permission of 755:

    ```
    .profile
    .login
    .cshrc
    .kshrc
    .logout
    ```

  - Verify that user masks are set in accordance with your site policy using the following command:

    ```
    # grep umask /usr/users/*/.*
    ```

    The system default mask is set to 022.
- Review the `/dev` directory and verify the following:
  - That special devices have the proper permission.
  - That access to devices such as `mem`, `kmem`, and `swap` are properly protected (440).
  - That terminal ports are readable only by the owner.
  - That users do not own any devices other than their terminal device and their printer.

- Verify that your modem authentication is functioning as intended.
- Use the following commands to verify that the same user name is not used for different UIDs, including between the local /etc/passwd file and NIS:

```
# ( ypcat passwd ; grep -v '^[-+]' /etc/passwd ) | \
   sort -t: -k 1,1 -k 3,3n -u | \
   awk -F: '{if (n == $1) {print p; print}; \
                                   n=$1; p=$0}' | \
   more
```

- Use the following commands to verify that no user names use the same UID, including between the local /etc/passwd file and NIS:

```
# ( ypcat passwd ;  grep -v '^[-+]' /etc/passwd ) | \
   sort -t: -k 3,3n -k 1,1 -u | \
   awk -F: 'BEGIN {u=-1} {if (u == $3) \
                 {print p; print}; u=$3; p=$0}' | \
   more
```

- Use both of the following commands to verify that all accounts have local or NIS passwords:

```
# sort -t: -n /etc/passwd | awk -F: '$2 == "" print'
# /usr/tcb/bin/edauth -g | sed -f sed_file | egrep -v \
                          ':u_pwd=[^:]|:u_istemplate:'
```

  The commands in sed_file are as follows:

```
: top
/:\\$/ {
N
b top
}
s/:\\/:/g
s/:[<tab><space>]*:/:/g
s/:[<tab><space>]*:/:/g
```

- Use the following command to validate all the hidden files on your system:

```
# find / \( -name '.*' ! -name . ! -name .. \) -print
```

- Use the following command to verify that no device files exist outside the /dev directory:

```
# find / \( -type b -o -type c \) -print
```

- Ensure that entries in the following startup scripts are appropriate and that the files are properly protected:

```
/sbin/inittab
/etc/init.d
/sbin/rc?.d        ? is the run level
```

- Ensure that the data saved in `/var/adm/crash` in the event of a system crash is accessible only to `root` and `adm` users.
- Using a password cracker program such as the public domain `crack` program, ensure that user passwords cannot be determined.
- Verify your site's physical security as described in Section E.5.
- Verify the permissions and ownership on the following directories are as listed:

Directory	Permission	Owner	Group
/	755	root	system
/bin	755	root	system
/dev	640	root or bin	system
/dev/null	666	root	system
/dev/ttys	666	root	system
/etc	755	root	system
/etc/rc.config	755	bin	bin
/etc/exports	644	root	system
/etc/passwd	644	root	system
/etc/resolv.conf	644	root	system
/etc/screend.config	755	root	system
/etc/sec	755	root	system
/home	555	root	system
/lib	755	root	system
/opt	755	root	system
/sbin	755	root	system
/sys	755	root	system
/tcb	755	root	system
/tmp	1777	root	system
/usr	755	root	system
/usr/bin	755	root	system
/usr/lib	755	root	system
/usr/ucb	755	root	system
/usr/ucb	755	root	system
/var	755	root	system

Directory	Permission	Owner	Group
/var/adm	755	root	system
/var/adm/crash	700	root	system
/var/adm/cron	755	root	system
/var/spool	755	root	system
/var/spool/cron	755	root	system
/var/spool/cron/atjobs	755	root	system
/var/spool/cron/crontabs	755	root	system
/var/spool/cron	755	root	system
/var/tcb	755	root	system
/var/tcb/audit	755	root	system
/var/tcb/bin	755	root	system
/var/tcb/files	755	root	system

E.8 Documents

The following documents will help you create and maintain a secure computing environment:

- *Site Security Handbook (RFC 1244)*. This handbook is the product of the Site Security Policy Handbook Working Group, a combined effort of the Security Area and User Services Area of the Internet Engineering Task Force. An online copy of this document is available at http://www.net.ohio-state.edu/hypertext/rfc1244/toc.html
- Tru64 UNIX *Installation Guide*
- Tru64 UNIX and *Installation Guide — Advanced Topics*
- *Trusted Computer System Evaluation Criteria*. U.S. Department of Defense, National Computer Security Center, DoD 5200.28-STD, December, 1985. This document, known as the *Orange Book* because of the color of its cover, is the U.S. Government's definitive guide to the development and evaluation of trusted computer systems. An online copy of the *Orange Book* is available at http://nsi.org/Library/Compsec/orangebo.txt
- *Password Management Guideline*. U.S. Department of Defense, (CSC-STD-002-85), April 12, 1985. This document, known as the *Green Book* because of the color of its cover, supports the *Orange Book* by presenting a set of recommended practices for the design, implementation, and use of password-based user authentication mechanisms. An online copy of the *Green Book* is available at

http://www.radium.ncsc.mil/tpep/library/rainbow/CSC-STD-002-85.html

The following documents will help you understand security concepts and procedures:

- *Computer Security Basics* - O'Reilly and Associates, Inc.
- *Practical UNIX Security* - O'Reilly and Associates, Inc.
- *UNIX: Its Use, Control, and Audit* - Contact the Institute of Internal Auditors Research Foundation at 249 Maitland Avenue, Altamonte Springs, Florida 32701-4201.

E.9 Tools

The following tools can help you maintain a secure environment:

- `crack` — A public domain password-checking program available at `ftp://ftp.cert.org/pub/tools/crack/`.
- `tripwire` — An integrity-monitor for UNIX systems. The `tripwire` software uses several checksum and signature routines to detect changes to files and to monitor selected items of system-maintained information. The program also monitors for changes in permissions, links, and sizes of files and directories. The `tripwire` package can be downloaded from `ftp://coast.cs.purdue.edu/pub/COAST/Tripwire/`.
- `COPS` — The Computer Oracle and Password System (COPS) package from Purdue University examines a system for a number of known weaknesses and alerts the system administrator to them; in some cases it can automatically correct these problems. The COPS package can be downloaded from `ftp://ftp.cert.org/pub/tools/cops/`.
- `SATAN` — SATAN is a tool that helps system administrators recognize several common networking-related security problems. It reports the problems without actually exploiting them. For each type of problem found, SATAN offers a tutorial that explains the problem and what its impact could be and what can be done about the problem. SATAN and several other security tools and documents are available at Wietse Zweitze Venema's Web site at `ftp://ftp.win.tue.nl/pub/security/index.html`.

The following script is an example of a tool you can create to extract login and logout information from the audit logs:

```
#!/usr/bin/ksh –ph

# Script to return summary of login/logout activities on the
# system since the last time it was run.
```

```
export PATH=/usr/sbin:/usr/bin:/usr/ccs/bin:/sbin

# where this script should run
Bdir=/var/adm/local
# where to find audit log files
Adir=/var/audit

Ofile="${Bdir}/lasttime"
Nfile="${Bdir}/newtime"
Afile="${Bdir}/lastdata"
Tfile="${Bdir}/lastmsg"

Events="-e trusted_event"

umask 077

# ensure the output format we need from date.
export LANG=C LC_ALL=C
export TZ=:UTC

if [ ! -f "${Ofile}" ]
then
 print 700101000001 > "${Ofile}"
 touch -t 197001010000.01 "${Ofile}"
fi

date +%y%m%d%H%M%S > "${Nfile}"

curfile=$(auditd -q)
auditd -dx
sleep 20       # give time for compression of the old log
while [ -f "$curfile" -a -f "$curfile".Z ] || [ -f "$curfile" \
                                    -a -f "$curfile".gz ]
do
    sleep 2 # wait some more
done

: > "${Afile}"

for af in $(find "$Adir" -name "auditlog.*" -newer "${Ofile}" \
                           -print | sort)
do
 audit_tool -b -t $(<"${Ofile}") -T $(<"${Nfile}") >> \
                     "${Afile}" -o -Q $Events "${af}" 2>/dev/null

 # the suppressed errors are for the {un,}compressed messages
done

TZ=:localtime
```

```
if [ –s "${Afile}" ]
then
 audit_tool –B –Q "${Afile}" > "${Tfile}"
 if [ –s "${Tfile}" ]
 then
  Mail –s 'login/out audit summary' root < "${Tfile}"
 fi
fi

mv –f "${Nfile}" "${Ofile}"
rm –f "${Afile}"
```

The following is the crontab entry for the above logging script:

```
0 9 * * * /var/adm/local/lreport
```

F

Enhanced Security in a Cluster

F.1 Overview of Security in a Cluster

The following discussion is based on a cluster consisting of Compaq systems running Tru64 UNIX Version 5.1A and TruCluster Server Version 5.1A.

A TruCluster Server cluster is a single security domain. Identification and authentication, access control lists (ACLs), and auditing are configured identically on each member by default, presenting a coherent interface to the user and the system administrator.

Because a single copy of the authentication files is shared among all cluster members, each user account is valid on all cluster members and a user can log in to the cluster without concern for which member accepts the connection. Identically configured ACL checking means consistent authorization and file access control; a user has the same access rights from every member. Clusterwide audit settings ensure a uniform capture of cluster activity.

F.2 Enabling Security Features in a Cluster

The following sections describe how to enable the enhanced security features in a cluster.

F.2.1 Access Control Lists

The Security Configuration icon on the Custom Setup menu of the `secconfig` utility contains a checkbox that enables or disables access control list support on all cluster members, under either base or enhanced authentication. ACL support determines the state (enabled or disabled) of access checking and ACL inheritance. Note that ACLs can be created, modified, deleted and examined regardless of the state of ACL support.

The NFS file systems require the `proplistd` daemon to support ACLs.

F.2.2 Audit

The Audit Configuration icon on the Custom Setup menu of the `auditconfig` utility configures audit options clusterwide. The kernel build performed by cluster creation automatically includes support for audit.

Audit configuration occurs in two steps. In the first, the parameters for the audit daemon, such as the name and location of the audit log file, are specified. In the second, a set of events to audit, called the audit mask, is selected. The configuration utility ensures that the same parameters and events are used on each cluster member. To maintain a single point of administration, the `auditconfig` utility stores this audit configuration information in the `/etc/rc.config.common` file.

F.2.3 Authentication

Both base and enhanced authentication are supported in a cluster. Base security, which uses the standard UNIX `/etc/passwd` and `/etc/group` authentication files, is the default configuration. Enhanced authentication, which moves passwords into an authentication database, is configured using the Security Configuration icon on the Custom Setup menu of the `secconfig` utility.

The ideal time to configure enhanced authentication for a cluster is before loading the TruCluster Server license and subsets and creating the cluster. You must select the enhanced security subsets (OSFC2SECxxx and OSFXC2SECxxx) during the installation. Enhanced authentication options are enabled clusterwide in the Custom or Shadow Password mode. (You can also enable ACL support from this utility.)

F.2.4 Distributed Logins and NIS

A cluster provides a common authentication environment to enable secure, distributed, highly available logins. The cluster can additionally function as a NIS master, slave, or client. (Note that all cluster members must play the same role in a NIS environment.)

As a NIS master, the cluster supports the NIS distribution of both standard user profiles from `/etc/passwd` and the enhanced user profiles available with enhanced security maintained in the protected password database. These enhanced user profiles can be distributed by NIS as the `prpasswd` map, in the same manner that `/etc/passwd` is distributed as the `passwd` map.

F.2.5 Configuring a NIS Master in a Cluster with Enhanced Security

To set up a cluster running enhanced security as a NIS master, perform the following procedure:

1. Set up enhanced security as previously discussed.
2. Load `/var/yp/src`, including `passwd` with specified accounts, as discussed in the Tru64 UNIX *Network Administration: Services* guide.

3. Set up the `prpasswd` map with one line per entry using `convuser -Mc`. See the `convuser`(8) reference page.
4. Set up NIS as described in the Tru64 UNIX *Network Administration: Services* guide. When running `nissetup`, select the `-S` security option of `ypbind` to bind the member to an authorized list of NIS servers and specify the cluster alias as one of these servers.
5. Modify the map maintenance scripts to support `prpasswd` maps as discussed in the Tru64 UNIX *Network Administration: Services* guide.

Notes

In domains where one or more nodes are running enhanced security and that mix Tru64 UNIX Version 5.1A and higher and DIGITAL UNIX Version 4.x systems, a Tru64 UNIX Version 5.1A or higer system must be the master. If none of the nodes are running enhanced security, a Version 4.x system can be the master.

The `dxaccounts` utility does not allow you to add a NIS account and change the user's security options at the same time. You must first create the account and then change the user's security options.

The `useradd` command fails unless the user's primary group is defined in the `/var/yp/src/group` map.

F.3 Authentication in a Cluster

Enabling enhanced security introduces a new daemon, `prpasswdd`. Two instances of the daemon, a parent and a child, execute on each cluster member. The parent is primarily responsible for starting or restarting the child. The child is responsible for writing changes to the authentication database. To eliminate lock contention in a cluster, only the daemon on the cluster member serving the `/var` mount point actually performs writes for all clients. The other `prpasswdd` daemons are in hot standby mode. If the cluster member serving the mount point and containing the active daemon fails, another member assumes both roles.

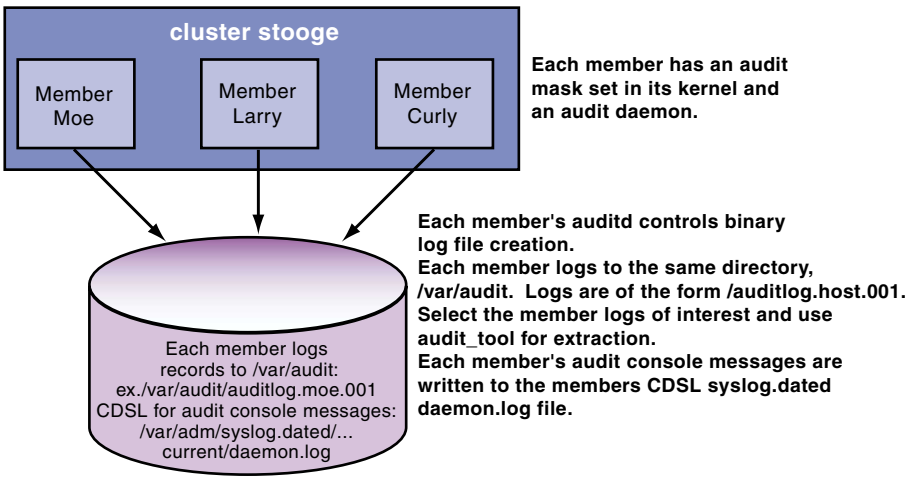
On a cluster acting as a NIS master with enhanced security, the `rpc.yppasswdd` daemon acts in the same fashion as the `prpasswdd` for the NIS `prpasswd` map.

F.4 Auditing in a Cluster

In a Version 5.1A cluster, no additional kernel rebuild is required to enable auditing because the DEC_AUDIT kernel option and the `/dev/audit` special file are automatically included on all cluster members in the initial cluster kernel build.

When auditing is enabled, an audit daemon (`auditd`) runs on each member of the cluster. Each audit daemon records event-specific audit log entries in a private audit log file, named by default `/var/audit/auditlog.{`*membername*`}.`*nnn*. Each audit log entry includes the hostname of the member on which it occurred, facilitating a merged display of the entries from multiple audit log files. Figure F–1 illustrates how audit data is gathered.

Figure F–1: Audit Data Flow in a Cluster



Each audit daemon writes its error or status messages to the local `/var/adm/syslog.dated/`*DATE*`/daemon.log` file, or optionally to a common audit console file.

The set of events to be audited, called the audit mask, is initially generated using the `auditconfig` utility. This utility specifies a common audit mask that is created clusterwide. Initial start up of the audit daemon uses information from the `/etc/rc.config.common` file. Thus, the same `auditd` command-line options and the same audit mask are used for each member.

On a running cluster, the `auditd` commands may be directed to every active daemon using the `auditd -cluster` option. For instance, `auditd`

`-cluster -w` displays the status of each member. Similarly, the `auditmask -cluster` option can change or display the audit mask on every active member.

While it is possible to specify different audit daemon parameters or different audit masks for the audit daemons in a cluster, this can be confusing and is not recommended.

The `audit_tool` utility can merge audit log files to present a clusterwide view of events sorted by forward time progression. Because host name information is recorded with each event, event origin can easily be determined.

F.4.1 Cluster Command Examples

This section provides examples of audit commands as they would be executed on a member system in a two member (`haydn` and `handel`), Version 5.1A cluster.

To get each member's `auditd` status, enter the following:

```
# auditd -cluster -w

Audit data and msgs:
  -l) audit data destination                = /var/audit/auditlog.handel.003
  -c) audit console messages                = syslog

Network:
  -s) network audit server status (toggle)   = off
  -t) connection timeout value (sec)         = 4

Overflow control:
  -f) % free space before overflow condition = 10
  -o) action to take on overflow             = ignore

cluster member haydn.zk3.dec.com auditd standard output:

Audit data and msgs:
  -l) audit data destination                = /var/audit/testauditlog.haydn.003
  -c) audit console messages                = syslog

Network:
  -s) network audit server status (toggle)   = off
  -t) connection timeout value (sec)         = 4

Overflow control:
  -f) % free space before overflow condition = 10
  -o) action to take on overflow             = ignore
```

To get each member system's audit mask status, enter the following:

```
# auditmask -cluster

!  Audited system calls:

!  Audited trusted events:
login                                   fail

!  Audstyle flags: exec_argp exec_envp login_uname obj_sel
```

```
**** cluster member haydn.zk3.dec.com standard output: ******
!  Audited system calls:

!  Audited trusted events:
login                                   fail

!  Audstyle flags: exec_argp exec_envp login_uname obj_sel
```

To determine what log file names the members are currently logging to, enter the following (entered from member `hayden`):

```
# auditd -cluster -q

/var/audit/auditlog.haydn.001

cluster member handel.zk3.dec.com auditd standard output:
/var/audit/auditlog.handel.001
```

To explicitly flush all the cluster member's kernel `auditd` buffers to get the latest audit records, enter the following:

```
# auditd -cluster -d
```

To print the login event for both members to `stdout`, enter the following:

```
# audit_tool -e login auditlog.handel.001 auditlog.haydn.001

ruid/euid:   0/0
pid:         525424        ppid: 525423              cttydev: (6,1)
event:       login
login name:  root
devname:     /dev/pts/1
...........
-- remote/secondary identification data --
hostname:    mk.zk3.dec.com
...........
char param:  argv=login -h mk.zk3.dec.com -p
char param:  Failed authentication
error:       13
ip address:  10.0.0.1 (haydn-mc0)
timestamp:   Mon May  3 15:54:18.19 1999 EDT


ruid/euid:   0/0
pid:         525424        ppid: 525423              cttydev: (6,1)
event:       login
login name:  (nil)
devname:     /dev/pts/1
...........
-- remote/secondary identification data --
hostname:    mk.zk3.dec.com
...........
char param:  argv=login -h mk.zk3.dec.com -p
char param:  Failed authentication
error:       13
ip address:  10.0.0.1 (haydn-mc0)
timestamp:   Mon May  3 15:54:26.44 1999 EDT


ruid/euid:   0/0
pid:         1049810       ppid: 1049809             cttydev: (6,2)
event:       login
login name:  root
```

```
devname:     /dev/pts/2
...........
-- remote/secondary identification data --
hostname:    mk.zk3.dec.com
...........
char param:  argv=login -h mk.zk3.dec.com -
char param:  Failed authentication
error:       13
ip address:  10.0.0.2 (handel-mc0)
timestamp:   Mon May  3 15:54:37.74 1999 EDT


ruid/euid:   0/0
pid:         1049810       ppid: 1049809              cttydev: (6,2)
event:       login
login name:  (nil)
devname:     /dev/pts/2
...........
-- remote/secondary identification data --
hostname:    mk.zk3.dec.com
...........
char param:  argv=login -h mk.zk3.dec.com -p
char param:  Failed authentication
error:       13
ip address:  10.0.0.2 (handel-mc0)
timestamp:   Mon May  3 15:54:43.14 1999 EDT


4 records output
6 records processed
#
```

F.5 Restrictions

The following restrictions apply to Tru64 UNIX Version 5.1A and TruCluster Server Version 5. 1A when it is run with enhanced security.

F.5.1 Upgrades

Upgrading from base to enhanced authentication in an existing cluster requires a full cluster reboot. The upgrade copies user accounts from `/etc/passwd` into `/var/tcb/files/auth.db`, removes passwords from `/etc/passwd`, and switches to a new security library. A new process authenticating a user name and password, such as a `telnet` session, uses the new library and accesses the new databases. However, an existing process, such as a `dtlogin` session or a locked CDE window, continues to use the original library. Because this library expects to access `/etc/passwd`, from which passwords have been removed, an existing process consistently encounters password verification failures. In particular, the console login window encounters this problem, which can create the erroneous belief that the root account is disabled. The cluster reboot prevents this situation.

F.5.2 Terminal Logging

Terminal logging is not available in a cluster.

When enhanced security is enabled, the system administrator can optionally enable terminal logging. When terminal logging is enabled, a terminal entry in the security terminal database, `etc/auth/system/ttys.db`, is updated with login information whenever a login occurs over the terminal. This logging is controlled by an option in the `secconfig` utility, or from the `d_skip_ttys_updates` field in the `/etc/auth/system/default` file. This setting is ignored in cluster members and terminal logging is not performed.

G

Division of Privileges

This appendix provides information about the division of administrative privileges using the DOP (division of privileges) utility.

G.1 Assigning System Administration Privileges

Without a division of privileges feature, only authorized `root` users (superusers) have access to many of the system administration programs. To split the responsibilities of administering systems, it is necessary to share the `root` password, a significant security risk. When an authorized person is removed from the list of authorized users, it is necessary to change the `root` account password.

The `dop` utility enables a `root` user to assign access to certain classes of administrative tasks to other users or groups of users. This enables a site to minimize access to the `root` account. Users or members of groups assigned a particular privilege can perform administrative tasks without knowing the `root` password. For example, a user granted the `AccountManagement` privilege can run the tasks listed under the Accounts branch of the SysMan Menu.

Administrative tasks (or actions) are organized into related groups and named for the particular privilege they confer to a user. For example, the `Security` privilege is extensive, and when granted to users other than `root`, it enables those users to run the following tasks:

`dopconfig`	Run `dop`, and grant privileges to other users.
`Dopaction`	Run from the Sysman Menu to grant privileges to other users.
`secconfig`	Configure the overall system security level.
`auditconfig`	Configure the audit environment on the system using the SysMan Menu interfaces.
`dxaudit`	Configure the audit environment on the system using the X11 compliant interface, Audit Configuration.

This is available from the CDE Application Manager —> Configuration menu.

This organization of tasks ensures that no matter what type of interface the privileged user invokes, he or she will have the authority to use it as if they were the `root` user.

In contrast, other privileges, such as `mailManagement`, are not extensive, and grant only access to the mail management task. However, before granting privileges to groups, you should use SysMan Menu to ensure that the members of a group are appropriate candidates for the privileges. If NIS is in use, you should also check the members of NIS groups before assigning a NIS group any privileges. Use the SysMan Menu Accounts option to review groups, and if necessary, to create new groups of privileged users.

To view a complete list of privileges, invoke `dop` as described in the following sections, which also describe the process of granting privileges to users and groups.

G.2 Invoking dop

You can invoke `dop` by the following methods:

- From the command line, type `dop` followed by the command options you require. For information on the `dop` command options, refer to the `dop(8)` reference page. The command line is mostly used for maintaining the privilege database, but it does also enable you to launch privileged actions directly from the command line, as described in Section G.3.
- From a command-line on a terminal, you can also invoke the character-cell SysMan Menu interface to `dop` using the `sysman dopconfig` command.
- From the SysMan Menu Security option, expand the menu option list and select the option titled Configure Division of Privileges (DOP). The SysMan Menu interface for `dop` can be run from a terminal, a personal computer (using Java), or an X11-compliant windowing environment. Refer to the *System Administration* guide for information on using the SysMan Menu.

  When `dop` is run as a graphical user interface or SysMan Menu option, online help is available for the interface. For information on using this interface, see Section G.5. Note that you can also write scripts to access `dop` features using the SysMan Menu command-line interface `sysman -cli`. For more information, refer to the *System Administration* guide and the `sysman_cli(8)` reference page.
- From the CDE Application Manager, select the Configuration folder and invoke `dop` by clicking on the DOP icon. Note that this invokes

the SysMan Menu and launches the X11 version of the interface. Refer to the *System Administration* guide for information on using CDE. When dop is run as a graphical user interface or SysMan Menu option, online help is available for the interface. For information on using this interface, see Section G.5.

G.3 Using the dop Command Line

You can maintain the dop database or attempt to run privileged actions (tasks or programs) using dop command-line options. The SysMan Menu option described in Section G.5 provides you with an easy way to look up privileges and actions; however it does not provide the dop database options available from the command line.

For a complete list of the command-line options, refer to the dop(8) reference page. Help on the dop command options can be obtained by typing dop -h.

The following sections provide some examples of using dop from the command line.

G.3.1 Launching Privileged Actions (Tasks)

You can attempt to run any action (a task or program) from the command line, and the dop utility verifies your access to the privilege against the privilege database as follows:

- If you are not root user, typing the command dop *action* prompts for the root password. For example:

  ```
  # dop users
  ```

- The -N option attempts to run the action without privileges. For example, to attempt to run the AccountManagement action users, enter:

  ```
  # dop -N users
  ```

- The -n option invokes a prompt asking if you want to run the command as yourself or as root user. You need the root password for the latter. To attempt to run the HostManagement action dxhosts unprivileged, type:

  ```
  # dop -n dxhosts
  ```

If you have the privilege, the action is launched. In this case, the user management interface of the SysMan Menu Accounts option is launched.

G.3.2 Administering the DOP Database

To administer the DOP database, you use certain command-line flags and options with the dop command. The database is a binary image to ensure

security, so only the dop command should be used to make changes. Only the root user is permitted to make changes to the database.

Administering the database involves adding, deleting, or modifying database entries and updating (writing) the binary file. Before you perform any of these tasks, you need to gather the following information:

1. Optionally, an *action* name — If you are adding *actions* to the database, you need to assign a name to it that enables you to easily recognize what program is being run by the *action*. For example, if you add /usr/bin/X11/xhost to the HostManagement privileges, you may want to name it hostmanager_cli to identify it as a command-line interface.
2. Required — The names of any privileges that you plan to change. To obtain a complete list of privileges you need to use the SysMan Menu DOP option, as described in Section G.5.
3. Required — The absolute paths to any programs that you intend to add to a privilege. For example, /usr/bin/X11/xhost. You also have an option to specify (or restrict) the type of user interfaces under which the action will be launched, such as X11 for CDE, or cli for command-line interface. Ensure that the program will run under any interfaces that you intend to specify. Programs that run under SysMan Menu will usually support the greatest range of interfaces.

Using the information you gathered and dop options, you can perform the following administrative tasks:

- Add new actions to the database, updating the binary image. For example, if you want to add the /usr/sbin/adduser utility to the AccountManagement privileges, use the following command:

  ```
  #dop -a AccountManagement adduser_script /user/sbin/adduser
  ```

  Note that this comand assigns the name adduser_script to the action and that name is used by privileged users to invoke the action. The dopaction editor (Manage DOP Actions) can also be used to add new actions and you will not have to worry about binary images.

  Using the method of viewing privileges described in Section G.5, you will see the new action added to the list of AccountManagement *actions* as follows:

  ```
  Priv AccountManagement is required by action(s)
  accounts
  users
  groups
  nis_users
  nis_groups
  adduser_script
  ```

- Delete existing actions from the database, updating the binary image. For example, if you want to select the `/usr/sbin/adduser` utility added in the previous example and remove it from the `AccountManagement` privileges, use the following command:

  ```
  # dop -d adduser_script
  ```

  The `dopaction` editor (Manage DOP Actions) can also be used to delete actions and you will not have to worry about binary images.

  Using the method of viewing privileges described in Section G.5, you will see that the action was removed from the list of AccountManagement actions as follows:

  ```
  Priv AccountManagement is required by action(s)
  accounts
  users
  groups
  nis_users
  nis_groups
  ```

- To write the binary image of the revised database, you have the following options:

`dop -w`	Writes the binary image without changing the source.
`dop -W`	Updates `actionlist` from the `dop` action file then writes binary image.

G.4 Defining and Managing New Actions

The Manage DOP Actions facility allows administrators to create new actions which include commands the actions execute as well as what privileges are required to run the actions. The Manage DOP Actions facility can be started from the Security branch of the SysMan Menu or from the command line with the following command:

```
# sysman dopaction
```

Caution

Compaq strongly recommends that users of the Manage DOP Actions facility *do not* modify the DOP actions that are supplied with the operating system. To do so might cause the system management facilities integral to the Tru64 UNIX system to fail. The Manage DOP Actions facility is provided to allow administrators to add and maintain new actions corresponding to tasks particular to their production environments.

G.5 Viewing or Modifying Privileges Using SysMan

The SysMan Menu option, Configure Division of Privileges (DOP), is located under Security. When you select this option, a window is displayed titled Configure DOP on *hostname*, where *hostname* is the name of the local system. This interface does not offer as many options as the `dop` command-line. Refer to the `dop`(8) reference page for more information on command line options, and see Section G.3 for examples of use.

The Configure DOP on *hostname* window enables you to view or modify the current assignment of privileges by adding new users and groups, or removing existing users and groups. The SysMan Privilege List: field contains a list of all the currently available privileges. The following are examples of some of the privileges on the list, and a brief description of what a privileged user or group can do:

1. `AccountManagement` — Enables privileged users to create user accounts and groups on the local system. If NIS is in use, it enables the privileged user to create NIS accounts and groups.
2. `EventManagement` — Enables privileged users to administer certain Event Management (EVM) services such as the SysMan Menu View events option or monitor.
3. `HostManagement` — Enables privileged users to perform host management tasks associated with the `dxhosts` graphical interface or `xhost` command.

To view the current assignments of any privilege, either double-click on a list entry (such as `PowerManagement`) or highlight an entry and press the Modify... button.

A second screen titled Configure DOP: Modify privilege <*privilege_name*> is displayed, where *privilege_name* is the list item you selected, such as `NetworkManagement`. This window contains the following information and options:

- Description — A brief description of the actions allowed under this privilege, and the list of actions that are allowed. For example, the following actions are displayed under `AccountManagement`:

  ```
  accounts
  users
  groups
  nis_users
  nis_groups
  ```

  This means that users who are granted the `AccountManagement` privilege can use any user account management interfaces, such as `dxaccount` or the SysMan Menu option Accounts - Manage local users.

- Specific users granted this privilege — A space-separated list of user names from the `/etc/passwd` file. Every user in this list is currently a privileged user. The field is blank if no users or groups have been granted this privilege.
- Specific groups granted this privilege — A space-separated list of groups from the `/etc/groupfile`. Every member of every group in this list is currently a privileged user. The field is blank if no users or groups have been granted this privilege.
- Browse... — These buttons enable you to browse a list of all authorized system users and groups as follows:
  - Browse Specific users — Displays an alphabetic scrolling list of all authorized users from the `/etc/passwd` file. If NIS is in use, you will be shown all NIS users who have access to the local system. Use the mouse pointer to double-click on any user name to select that user.
  - Browse Specific groups — Displays an alphabetic scrolling list of all groups from the `/etc/group` file. If NIS is in use, you will be shown all NIS groups who have access to the local system. Use the mouse pointer to double-click on any group to select that group.

    Any authorized member of that group will be granted the privilege. You may want to create your own groups of privileged users so that you can administer privileges more easily. Refer to the *System Administration* guide for information on creating groups.

You can modify the existing privilege assignments, adding or removing existing users and groups, as follows:

1. In the window titled Configure DOP on *hostname*, select the required privilege and press the Modify... button.
2. In the window titled Configure DOP: Modify privilege <*privilege_name*>, change the assignments as follows:

   a. To administer users, type (or delete) user names from the space-separated list. Use the Browse... option button to view a list of current system users.

   b. To administer groups, type (or delete) group names from the space-separated list. Use the Browse... option button to view a list of current groups.

   c. Press OK to update the DOP database and exit from the window, or press Cancel to abort the operation. You will be returned to the previous window, where you can select another privilege or exit and return to the SysMan Menu.

Refer to the online help for more information on the options.

Glossary

absolute pathname
A pathname that begins at the root directory; a pathname that always begins with a slash (/). For example, `/usr/games` is an absolute pathname. Also called a full pathname.

Access ACL
The formal name of the ACL that is checked for access decisions on an object.

ACL (access control list)
An optional extension of the traditional UNIX permission bits, which gives the user the ability to specify read, write, and execute permissions on a per user and per group basis.

See also *Access ACL*, *Default ACLs*

administrator
This document uses the term "administrator" in a generic sense to refer to any user involved in the security operation of the system.

audit event
An event that is monitored and reported by the audit subsystem. Events include system events, application events, and site-definable events. An event can be any command, system call, routine, or program that runs on the system.

audit ID (AUID)
An ID that is created at login time and that is inherited across all processes.

auditing
The recording, examining, and reviewing of security-related activities on a trusted system.

BASE security
The traditional security that is delivered on BSD UNIX systems. BASE security consists of user authentications based on the `/etc/passwd` file. A nontrusted Tru64 UNIX system has BASE security.

BSD (Berkeley Software Distribution)
A UNIX software release of the Computer System Research Group of the University of California at Berkeley –- the basis for some features of Tru64 UNIX.

Default ACLs
The ACLs associated with directories. These two types of ACLs (default access ACL and default directory ACL) determine what ACLs are given to files and subdirectories created in a directory.

discretionary access control (DAC)
The traditional UNIX form of file permissions set with the `chmod` command.

effective user ID (EUID)
The current user ID, but not necessarily the user's ID. For example, a user logged in under a login ID may change to another user's ID. The ID to which the user changes becomes the effective user ID until the user switches back to the original login ID.

enhanced passwords
Passwords with the enhanced attributes made available by the enhanced security option. Enhanced passwords are stored in the `prpasswd` file and are sometimes refered to as extended, protected, or shadowed passwords.

ENHANCED security
The optional security feature that supplement BASE security. Enhanced security consists of enhanced password profiles.

See also *enhanced passwords*

entity
The security integration architecture (SIA) uses the term entity to mean a user, program, or system that can be authenticated. The entity identifier is the user ID (UID).

ER (external representation)
A POSIX-compliant ASCII representation of an ACL used for presentation to the user.

See also *IR (internal representation)*

evaluation criteria
The *Trusted Computer System Evaluation Criteria (TCSEC)*. The enhanced security features in the Tru64 UNIX system have been designed to meet this criteria.

IR (internal representation)
A binary representation of an ACL used by the ACL library routines.

See also *ER (external representation)*

login spoofing program
Any program that represents itself as a `login` program to steal a password. For example, a spoofing program might print the login banner on an unattended terminal and wait for input from the user.

operator
The person responsible for the day-to-day maintenance of a system, including backups, line printer maintenance, and other routine maintenance tasks.

See also *system administrator*

process ID (PID)
A unique number assigned to a process that is running.

process
A unit of control of the operating system. A process is always executing one program, which can change when the current program invokes the `exec()` system call. A process is considered trusted when its current program is trusted.

See also *program*

program
A set of algorithms designed, compiled, and installed in an executable file for eventual execution by a process. A program is considered trusted when it upholds the security policies of the system.

See also *process*

PPID (parent process ID)
The process ID of the parent or spawning process.

root
The login name for the superuser (system administrator).

root directory
The name applied to the topmost directory in the UNIX system's tree-like file structure; hence, the beginning of an absolute pathname. The root directory is represented in pathnames by an initial slash (/); a reference to the root directory itself consists of a single slash.

root file system
The basic file system, onto which all other file systems can be mounted. The root file system contains the operating system files that get the rest of the system to run.

security attributes
The parameters used by the trusted computing base (TCB) to enforce security. Security attributes include the various user and group identities.

SIA (Security Integration Architecture)
The Security Integration Architecture isolates the security-sensitive commands from the specific security mechanisms, thus eliminating the need to modify them for each new security mechanism.

See also *vouching*

site-defined events
Audit events that are created by application software (that is, not the operating system).

spoofing program
See *login spoofing program*

system administrator
The system administrator is responsible for file system maintenance and repair, account creation, and other miscellaneous administrative duties.

TCB (trusted computing base)
The set of hardware, software, and firmware that together enforce the system's security policy. The Tru64 UNIX TCB includes the system hardware and firmware as delivered, the trusted Tru64 UNIX operating system, and the trusted commands and utilities that enforce the security policy. The operating system and other software distributed with the trusted Tru64 UNIX system satisfy security requirements.

Traditional security
See *BASE security*

triviality checks
Checks performed on passwords to prevent the use of easily guessed passwords. Triviality checks prevent the use of words found in the dictionary, user names, and variations of the user name as passwords.

Trojan horse
Any program that when invoked by a user steals the user's data, corrupts the user's files, or otherwise creates a mechanism whereby the Trojan horse planter can gain access to the user's account. Viruses and worms can be types of Trojan horses.

See also *virus*, *worm*

virus
A computer program designed to insinuate itself into other programs or files in a system and then to replicate itself through any available means (disk file, network, and so forth) into other similar computers, from which it can attack yet more systems. Viruses are designed with the object of damaging or destroying the "infected" programs or systems and are often

programmed to become destructive at a specific time, such as the birthday of the virus's programmer.

See also *Trojan horse* , *worm*

vouching

A technique that allows a security mechanism to trust the authentication process of a previously run security mechanism. This feature is implemented by the Security Integration Architecture (SIA).

worm

A computer program designed to insinuate itself into other programs or files in a system and then to replicate itself through any available means (disk file, network, and so forth) into other similar computers, from which it can attack yet more systems. Worms are designed with no serious intent to do damage, but they are harmful because they occupy resources intended for legitimate use.

See also *Trojan horse* , *virus*

Index

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W

X