Digital UNIX

Security

Order Number: AA-Q0R2D-TE

March 1996

Product Version: Digital UNIX Version 4.0 or higher

This manual describes how to operate the Digital UNIX system in a secure manner. It includes traditional UNIX security procedures and operation with the optional enhanced security subsets installed. The Security Integration Architecture (SIA) and access control lists (ACLs) are also documented in this manual. This manual has information for users, system administrators, and programmers.

Digital Equipment Corporation
Maynard, Massachusetts

Contents

2 Getting Started

3 Connecting to Other Systems

4 DECwindows Environment

5 Using ACLs

Part 2: Administrator's Guide to Security

6 Introduction for Administrators

7 Setting Up the Trusted System

8 Creating and Modifying Secure Devices

9 Creating and Maintaining Accounts

10 Administering the Audit Subsystem

11 Administering ACLs

12 Ensuring Authentication Database Integrity

13 Security Integration Architecture

14 Trusted System Troubleshooting

Part 3: Programmer's Guide to Security

15 Introduction for Programmers

16 Trusted Programming Techniques

17 Authentication Database

18 Identification and Authentication

19 Audit Record Generation

20 Using the SIA Interface

21 Programming With ACLs

A File Summary

B Auditable Events and Aliases

C Interoperating with and Migrating from ULTRIX Systems

D Coding Examples

E Symbol Preemption for SIA Routines

Glossary

Examples

Figures

Tables

About This Manual

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

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.

New and Changed Features

This release of Digital UNIX adds access control lists (ACLs) to improve the security capabilities of the operating system. Chapters in the User, Administration, and Programming sections are added to document this new feature. Several reference pages that document the ACL commands and routines are also available on line. Use the `man -k acl` command to get a listing of the ACL-specific reference pages.

Object selection and deselection features are added to the audit subsystem. See Section 10.3.2 for more information.

Several authentication programming interfaces are added and changed in this release. The new APIs allow the authentication data structures to be extended. See Section 18.1 for more information.

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

This manual has 21 chapters, 5 appendixes, a glossary, and an index:

Chapter 1	Introduces the enhanced security features of the Digital UNIX system from a user's point of view and defines the areas in which trusted Digital UNIX 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 DECwindows Motif features that enhance the security of a workstation. This chapter does not explain how to use DECwindows.
Chapter 5	Describes the ACL (access control lists) features of system and how users can most effectively use them.
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 (or the DECwindows `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.
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 Digital UNIX 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 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 interfaces.

Chapter 21	This chapter provides the programmer with the information needed to use ACLs (access control lists) in applications that run on Digital 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	Explains the issues encountered when moving applications and accounts from ULTRIX systems to Digital UNIX systems.
Appendix D	Provides the programmer with extended coding examples for trusted Digital UNIX systems.
Appendix E	Explains the naming convention used to keep Digital UNIX compliant with ANSI C.

Related Documentation

The following documents provide additional information about security issues in the Digital 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

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

Trusted Computer System Evaluation Criteria (TCSEC or *Orange Book*)

Password Management Guideline (*Green Book*)

A Guide to Understanding Audit in Trusted Systems

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

Information Technology Security Evaluation Criteria (ITSEC).

The printed version of the Digital UNIX documentation set is color coded to help specific audiences quickly find the books that meet their needs. (You can order the printed documentation from Digital.) This color coding is reinforced with the use of an icon on the spines of books. The following list describes this convention:

Audience	Icon	Color Code
General users	G	Blue
System and network administrators	S	Red
Programmers	P	Purple
Device driver writers	D	Orange
Reference page users	R	Green

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

The *Documentation Overview, Glossary, and Master Index* provides information on all of the books in the Digital UNIX documentation set.

Reader's Comments

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

You can send your comments in the following ways:

- Fax: 603-881-0120 Attn: UEG Publications, ZK03-3/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`

- Mail:

  Digital Equipment Corporation
  UEG Publications Manager
  ZK03-3/Y32
  110 Spit Brook Road
  Nashua, NH 03062-9987

  A Reader's Comment form is located in the back of each printed manual.

The form is postage paid if you mail it in the United States.

Please include the following information along with your comments:

- The full title of the book and the order number. (The order number is printed on the title page of this book and on its back cover.)
- The section numbers and page numbers of the information on which you are commenting.
- The version of Digital UNIX that you are using.
- If known, the type of processor that is running the Digital UNIX software.

The Digital 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 Digital technical support office. Information provided with the software media explains how to send problem reports to Digital.

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 and Korn 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).
Menu→Option→ Submenu Option	The right arrow indicates an abbreviated instruction for choosing a menu option or submenu option. The following example means pulldown the Modify menu, move the pointer to pull down the Image submenu, and choose the Clear option: Choose Modify→Image→Clear

Part 1: User’s Guide to Security

Introduction for Users 1

The Digital UNIX operating system is delivered with an enhanced security optional subset. When this subset is installed and configured, the system is referred to as a trusted system. The Digital UNIX enhanced security features result in a trusted system designed 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 Digital 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 discretionary controls provided by the system. This chapter explains system capabilities and user responsibilities.

1.1 Enhanced Security Features

The Digital UNIX system without the enhanced security subset installed provides traditional UNIX security, as described in the Digital 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 extends traditional security.

1.1.1 Login Control Enhancements

Enhanced security provides the following features for login control:

- 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 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

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 features of a trusted Digital UNIX system is that the administrator can use the audit subsystem to hold users accountable for their actions. The audit subsytem records 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 recreate past events that affect the security of their accounts and data.

The audit feature is discussed in detail in Chapter 10.

1.2 How to Determine if Enhanced Security Is Installed and Running

If you are not sure if the optional, enhanced-security features are installed on your system, you can check as follows:

```
$ ls -l /usr/.smdb./OSFC2SEC4??.lk
-rw-r--r--  1 root  system  0 Nov  8 11:02 \
                                /usr/.smdb./OSFC2SEC400.lk
```

The presence of the lock file (`OSFC2SEC400.lk`) indicates that the enhanced security subset is installed (but not necessarily running) on your system. If the subset is not installed, you will receive a ''not found'' message.

To determine if the installed enhanced security is running on your system, enter the following command:

```
$ /usr/sbin/rcmgr get SECURITY BASE
ENHANCED
```

If the string ''ENHANCED'' is returned, enhanced security is running. If the string ''BASE'' is returned, enhanced security is not running.

1.3 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. Once stamped, the AUID cannot be changed.

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.

1.4 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 (file and directory permissions) to protect your data from disclosure or destruction.
- Report all suspect activity to the ISSO, so that past events can be analyzed through the audit trail.

A trusted Digital 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.

Getting Started 2

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 Digital UNIX 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 Digital 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 Digital UNIX is similar to the procedure for nontrusted Digital UNIX systems. This section describes the general process. See the `login`(1) reference page for details.

On a trusted Digital UNIX 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 Digital UNIX 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 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 ISSO 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)
- Effective and real user IDs (EUID, RUID)
- Effective and real group IDs (EGID, RGID)
- Supplementary groups

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 Digital UNIX 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 locked. 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. If the terminal is 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 be explicitly locked. 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 ISSO has to enable it again before you can log in.

2.2 Setting Your Password

A trusted Digital 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 2.2.1.)
- 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 ISSO 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 ISSO 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 letter only):
        pronounceable passwords generated for you (g) ?
        a string of characters generated (c) ?
        a string of letters generated (l) ?
        to pick your password (p) ?


Enter choice here:
```

If you enter `p`, 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 "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 ISSO to unlock 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 Digital 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 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 ISSO 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 ISSO 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 ISSO to have it changed.

2.6 Problem Solving

The trusted Digital UNIX's mechanisms may be somewhat unfamiliar if you are accustomed to a nontrusted Digital 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 Digital 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 ISSO 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 locked; only the ISSO can unlock your account. You must change your password before using the system again after the ISSO unlocks 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 ISSO has changed it for you. If this message appears when you do not expect it, see your ISSO.

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. Programs that manage to survive the user's logout can still 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 it will be given some of the characters that are also requested by the `getty` and `login` programs for the new user session.

The Digital 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 Digital UNIX systems, trusted Digital 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 Digital UNIX system default `umask` is 077. If unauthorized users try to access such a file, they will only 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 Digital 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 If You Cannot Log In

There are a number of reasons why a login attempt can fail on a trusted Digital 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 ISSO.

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 ISSO, 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 ISSO.
- Your account is disabled. See your ISSO to have your account unlocked. 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 disabled by the ISSO.
- Your password has expired. See your ISSO to have your account unlocked. You can change your password during the next log in.

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

Connecting to Other Systems 3

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 Digital 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 Digital 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 Digital 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.

DECwindows Environment 4

This chapter discusses DECwindows 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, your workstation determines which hosts are authorized to access its display. Every user who can log in to an authorized host has the following kinds of access to your workstation:

- Read

  Users can read the contents of one or more windows on your workstation. When you press a key on your keyboard a character representing the key appears on your workstation screen. Thus, you can see what you type on your screen. Any user on a host that is authorized to access your display could divert your keystrokes to another workstation display. An unscrupulous user could capture and display keystrokes (including your password) on another system.

- Write

  Users on authorized hosts can send simulated keystrokes to your workstation display. Your workstation software treats the keystrokes the same whether you type them from your keyboard or an application program sends them. Users on authorized hosts can send commands to your workstation — and every command is executed under your user login and password. For example, any user on an authorized host could delete all the files in your home directory tree.

- Copy

  Users on authorized hosts can capture a snapshot of any one of your windows or your entire workstation screen, without your knowledge. This snapshot is a static picture of the contents of your display. In general, if you can see it on your display, any user on an authorized host can see the same thing.

4.2 Controlling Network Access to Your Workstation

Controlling access to your workstation display is the key to creating a secure workstation environment. Your workstation keeps an access control list (ACL), which names the hosts on a network that can access its display. This list is a combination of a system list that your security administrator creates and a personal workstation list that you create.

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

There are three ways to designate which hosts can access your workstation display:

- The system ACL
- The workstation ACL
- The X authority file utility

4.2.1 System Access Control List

Your security administrator can authorize a host to access a workstation's display by adding the host name to a systemwide authorization file called `/etc/X*.hosts`. The asterisk (*) refers to the number of the workstation display that the hosts listed in the file can access. The standard display number is 0 (zero). Hosts that are not listed in this file cannot access your workstation display. When shipped with your system, the `/etc/X*.hosts` file is empty, which means that only your workstation (the local host) can access its display.

4.2.2 Workstation Access Control List

Your workstation ACL can allow hosts access to your workstation display even though the system ACL does not. You can thus explicitly authorize other users or yourself, when you are logged in from another host, to display DECwindows applications and programs on your workstation.

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.

Take the following steps to authorize other users to use your workstation display:

1. Select the Session Manager window.
2. Select the Security... option from the Options menu. The Security Options box is displayed on the screen. Type the host name you want to authorize.
3. Click on the Add button. The host name is added to the Authorized hosts box.
4. Click on the OK or Apply button.

To remove a host name for the current session:

1. Click on the name you want to remove.
2. Click on the Remove button.
3. Click on the OK or Apply button.

Users logged in to the host you remove will no longer have access to your workstation for this session. However, the system ACL is checked each time you start a session. Thus, removing a host is temporary if the host is listed in the `/etc/X*.hosts` file.

4.2.3 Storing the Workstation Access Control List

The changes you make to your workstation ACL remain in effect only for the current session unless you save them. You can save the changes you make during a session from the Customize menu in the Session Manager window. When you save the changes you make during a session, the hosts listed in the Customize Security box are stored in a file called `.Xdefaults`, in your home directory. Each time you start a new session, the workstation checks the `/etc/X*.hosts` system file as well as the `.Xdefaults` file to determine its ACL.

Any user who can edit the `.Xdefaults` file could modify the ACL for your workstation display. If that happens, the new list of authorized hosts would become effective the next time you start a session.

Therefore, check your file permissions. Your home directory should deny read, write, and execute access to *other*, and write access to *group*. The permissions on the `.Xdefaults` file should deny all access to *group* and *other*. Use the `chmod` command to change the permissions:

```
$ chmod 750 $HOME
$ chmod 600 .Xdefaults
```

4.2.4 Using the X Authority File Utility

The `xauth` program allows you to run client applications on other workstations that do not share their home directory. You use the `xauth` program to edit and display the authorization information used in connecting 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 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 Keyboard Input

DECwindows includes a secure keyboard mode that directs everything you type on the workstation keyboard to a single, secure window. All keyboard input is directed to the secure window, even if you have selected another window for input focus. In secure keyboard mode, keyboard input is read only by the application that created the window.

Secure keyboard mode is useful for protecting sensitive information, like your password, because it prevents users from running applications that might capture your keystrokes. Setting secure keyboard mode in a window prevents users on hosts that are authorized to access your workstation display from reading any keyboard input from that window. For example, if you have a root account on your workstation, always set secure keyboard mode before using `su` and typing your root password. You can set secure keyboard mode by selecting the Secure Keyboard item from the Commands menu in a DECterm window.

If hosts are authorized to access your workstation display, users on those hosts can still copy the contents of your display at any time. When you use the `su` or `passwd` command and type your password, the password does not appear on the screen. Therefore, a static copy of your display will not reveal your password. A static copy could, however, reveal the contents of a sensitive file displayed on your screen. If you are working on sensitive files, do not authorize any host to access your display.

After you select the Secure Keyboard item, the window appears in reverse video, and the toggle button next to the Secure Keyboard item appears highlighted to indicate that security mode has been set.

When you change a secure window to an icon, the secure keyboard mode is turned off. If you want security to be on, you must turn it on again when you change your icon back to a window.

You can create only one secure window at a time. If you try to create a second secure window, you will hear a beep, reminding you that secure keyboard mode has been set for another window. If you hear a beep when you try to set secure keyboard mode, but have not set that mode in any other window on your screen, some other application must have set the mode. If this happens, check with your security administrator to find out which application may have set this mode.

4.4 Blocking Keyboard and Mouse Information

By default, DECterm windows block keyboard and mouse information sent from another computer. This means that users on another system cannot send simulated keystrokes or mouse clicks to your workstation. This security feature prevents unauthorized users from sending potentially destructive commands to your workstation when it is idle.

The ability of a DECterm window to block information sent from another host is set by a resource called `allowSendEvents`, which is set to FALSE in the `.Xdefaults` file. Each time you begin a session, DECwindows uses the values in this file to control the appearance and other characteristics of window displays on your workstation.

The following example shows a line in the `.Xdefaults` file that sets the `allowSendEvents` resource FALSE, thus blocking users logged in to other host systems from sending keyboard or mouse information to any window that you create.

```
Dxterm*allowSendEvents:  false
```

Leave the `allowSendEvents` value set to FALSE to prevent unauthorized users from sending input into your DECterm window and executing commands under your user name.

An application that opens its own window (not a DECterm window) might not block simulated keystrokes from your display. Therefore, if you are running such an application, check your ACL and remove any hosts that are authorized to access your display before working on sensitive files. If you must authorize a host to access your display (for example, to run a remote application), remember to set secure keyboard mode before using the `passwd` or `su` commands and typing your password.

4.5 Pausing Your Workstation

In a DECwindows environment, you can pause your current session. This locks your workstation without ending your session. Your screen is cleared, and the system displays the Pause screen. You can resume your session any time without recreating your screen environment.

To put your current session on hold, choose the Pause menu item from the Session menu. Your screen is cleared and the Continue Session box is displayed. To continue your session,: type your password then click on the OK button or press Return.

Once your password is verified, your session resumes.

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.

Using ACLs 5

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

The Digital UNIX ACLs are based on the POSIX P1003.6 Draft 13 standard. The ACL API (Application Programming Interface) may change as the P1003.6 standard is finalized.

5.1 Traditional Discretionary Access Control

Discretionary access control (DAC) is a means of restricting access to objects based on the access permissions attached to the objects. The controls are discretionary in the sense that a subject with a certain access permission is capable of passing that permission (perhaps indirectly) on to any other subject. If you own a file, for example, you can allow other users to read it or write to it by changing its access permissions.

The untrusted Digital UNIX file protection model, although simple and effective, does not meet the DAC requirements for trusted systems. Many environments require more granular control over file access than that provided by the Digital UNIX discretionary protection.

See the *Command and Shell User's Guide* for a complete description of discretionary access control.

5.2 An Overview of ACLs

To provide access granularity down to a single user, Digital UNIX objects 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 file systems that support property lists. An ACL allows users to specify exactly how they want their files protected.

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

- With separate access control specifications for each user and group. 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 all user and group permissions to be automatically specified upon object creation. 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 a default ACL for a directory, and it is inherited by files and subdirectories when they are created.

The following commands display and modify ACLs:

`setacl`	Changes the ACLs on files and directories.
`getacl`	Lists the ACLs on files and directories.

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

5.3 States of the ACL System

The system administrator can enable and disable the ACL subsystem on your machine. When the ACL subsystem is 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 file system objects and the library interfaces are still present. However, the following five ACL features are only available when ACLs are enabled:

- Inheritance
- Access checking
- Validation of ACLs being set
- Caching of ACL attributes
- ACL and `chmod` interactions

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

5.4 Setting an ACL

Setting an ACL is accomplished using the `setacl` command. The file must reside on a file system that supports the property list.

5.5 Default ACLs

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 ACL, if it exists, determines the ACLs created for descendents of the directory.

The default directory ACL allows the owner or a privileged user to associate an ACL with a directory that is to be inherited as an access ACL when a subdirectory is created. The default directory ACL will also be inherited as the default directory ACL by the new directory.

The default access ACL allows the owner or a privileged user to associate an ACL with a directory that is to be inherited as an access ACL when an object is created within the directory. If the object being created is a directory and a default directory ACL exists on the parent directory, it is inherited as the access ACL and not the default access ACL. The default access ACL is inherited as the default access ACL for any subdirectory created

5.6 Viewing an ACL

An ACL is viewed by using the `getacl` command. If there is no ACL associated with the object, the standard UNIX permission bits are shown in the ACL format.

5.7 Access Decision Process

Access decisions for objects are made as follows:

1. If the process has the superuser privilege, access to the object is granted.
2. If ACLs are not enabled, or they are enabled but there is not an ACL associated with the object, the traditional UNIX permission checks are used.
3. The ACL is checked as follows:
   a. If the process is the owner of the object, the permissions in the owning `user::` entry are granted.
   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.
   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 permissions of the `group::` entry are granted.
   d. If the GID or supplementary groups of the process match any of the `group:` entries, the union of the permissions of all matching entries is granted.

e. The permissions in the `other:` entry are granted.

Note that the permissions on the object are the permissions allowed by the first matching `user` or `group` entry or the `other` entry.

5.8 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 group or user ID or name
- A permission specification

In the external representation of an ACL, the fields are separated by colons (:). Entries in the qualifier field are separated by commas (,). The following example shows typical ACL entries:

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

The keywords and qualifiers are 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) corresponds to the user permission bits. The qualifiers for a `user` entry can be user names or numeric UIDs; such an entry defines the permissions of these users. An ACL must contain an owning user entry and can contain any number of qualified `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) corresponds to the group permission bits. The qualifiers for a `group` entry can be group names or numeric GIDs; such an entry defines the permissions of members of these groups. An ACL must contain an owning group entry and can contain any number of qualified `group` entries.

`other` The `other` entry defines the permission of all users who are not identified in a `user` or `group` entry. This entry corresponds to the other permission bits. An ACL has only 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.

Because the discretionary check against the ACL requires a match against one of the ACL entries in order for access to succeed, an object with an ACL having no entries that match the process's access request denies all accesses unless the requesting process has the ability to override discretionary access control checks.

Table 5-1 illustrates and explains typical ACL entries.

Table 5-1: 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.

5.9 ACL Initialization

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 `moun` 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. In addition, the file or directory inherits ACLs from its parent directory. A file only has one ACL associated with it, an access ACL. A directory can have three, an access ACL, a default access ACL, and a default directory ACL. The default ACLs determine what ACLs are inherited by the descendents of the directory, as follows:

- If the parent directory does not have a default access ACL, the access ACL on new files in that directory is initialized from the traditional permission bits.
- If the parent directory does not have any default ACLs, the access ACL on new subdirectories in that directory is initialized from the traditional permission bits. No default ACLs is set.
- If the parent directory has a default access ACL, the access ACL on new files in that directory is the same as the default access ACL of the parent.
- If the parent directory has a default access ACL and does not have a default directory ACL, the access ACL and default access ACL on new subdirectories in that directory are both set to the default access ACL of the parent. No default directory ACL is set.
- If the parent directory does not have a default access ACL, but it does have a default directory ACL, the access ACL and default directory ACL on new subdirectories in that directory are set to the default directory ACL of the parent. No Default Access ACL is set.
- If the parent directory has both a default access ACL and a default directory ACL, the access ACL and default directory ACL on new subdirectories in that directory is set to the default directory ACL of the parent. The default access ACL on new subdirectories in that directory is set to the default access ACL of the parent.

Some examples of ACL initialization follow:

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

  ```
  % setacl –d –u 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:

```
% setacl –D –u 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:

```
% setacl –D –u user:jdoe:rw– foo
% setacl –d –u 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 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 ACLs 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.

5.10 Protecting Objects with ACLs

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.

5.10.1 ACLs and the ls Command

In trusted Digital UNIX, as in traditional systems, the `ls -l` command displays the access allowed for the owning-user, the owning-group, and others. For the owning-user and for others, the permission displayed by `ls -l` and the permissions displayed by the `getacl` command are identical. The group permissions displayed by `ls -l` are the maximum permissions allowed for the owning group and for any user or group identified in a qualified `user` or `group` ACL entry. A given user or member of a group can have more restrictive permissions.

For example, assume that the following `group` and `user` entries are set in

an ACL:

```
user::rwx
group::r-x
user:fred:r--
user:chen:rw-
group:nosy:---
other::---
```

The owning group has read and execute permission, and the user `chen` has read and write permission. The `ls -l` command displays the following permissions for a file with this ACL:

```
-rwxr-x---
```

5.10.2 Using the setacl Command

The `setacl` command is used to modify, add, and remove entries from existing ACLs.

Follow these rules for using `setacl`:

- Specify the new attribute (that is, the ACL entries to be applied to the existing ACL, preceded by the appropriate command option, if any), followed by a list of objects to change.
- Specify ACL entries between commas, if entered directly on the command line, or on separate lines if listed in an ACL input file.

The examples that follow show the `setacl` command line with the options, input arguments, and results of applying `setacl` in text format.

- To set an ACL entry denying all access to the object `private` for user john, enter the following:

  ```
  % setacl -u user:john:--- private
  ```

- To set ACL entries allowing read and write access to the object `private` for members of the `finance` and `marketing` groups, enter the following:

  ```
  % setacl -u group:finance:rw-,group:marketing:rw- private
  ```

- Suppose the current ACL on the `ffff` file contains these entries:

  ```
  user::r--
  user:jean:r--
  group::r--
  other::---
  ```

You can update the ACL with the following command:

```
$ setacl –u \
  group:proj1:r––,user:jean:rw–,group:finance:r–– ffff
```

The resulting entries are as follows:

```
user::r--
user:jean:rw-
group::r--
group:proj1:r--
group:finance:r--
other::---
```

The `user:jean` permissions overwrite the existing permissions for the matching `user` entry. Both input `group` entries that do not match any of the existing entries in the existing ACL are added to the ACL.

- Suppose an ACL is as follows:

```
user::r--
group::r--
user:jean:rw-
group:proj1:r--
group:finance:r--
other::---
```

Use the following command to add a new permission:

```
$ setacl  –u  user:jean:rw–  ffff
```

The resulting ACL entries are:

```
user::r--
group::r--
user:jean:rw-
group:proj1:r--
group:finance:r--
other::---
```

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.

5.10.3 Using the getacl Command

The `getacl` command lists the ACL on a file in a manner similar to the `ls` command. For all regular file name arguments, `getacl` lists the ACLs of the files. The `–d` flag lists the default ACLs. The following is an example of

`getacl` usage:

```
$ getacl share_file

#
# file: shared_file
# owner: mary
# group: marketing
#
user::rw-
group::r--
other::---
```

The first user and group entries refer to the owning user and owning group, respectively.

5.11 Maintaining ACLs on Your Objects

The ACL programs are an extension to untrusted Digital UNIX systems and are defined by POSIX and System V compatible UNIX programs. All users should take extra precautions to ensure that ACL-protected objects remain protected (or are reprotected) after manipulation by a UNIX program.

When a program manipulates an object by creating a new version, removing the existing version, and then renaming the new version, the object's ACL is lost or replaced with the access ACL of the directory where the new version is created. When an object is renamed it retains the ACL (or lack of ACL) with which it was created. When a program modifies an object in place (that is, rewrites, appends, or updates the object without actually deleting it), the ACL protection is retained. ACLs are preserved as long as the object is not removed or the ACL is not explicitly deleted.

Until UNIX variants conform to a standard representation for ACLs, and the base utilities are converted to preserve ACLs from object to object, 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

Digital 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.

5.12 ACLs and the emacs Editor

After editing a file with the `emacs` editor, the new `emacs` copy of the file does not have the original ACL, but rather the default directory ACL.

The problem is that the `emacs` editor renames the old file prior to writing out the new text. This causes the following to happen:

- The ACL is associated with the backup file
- The default directory ACL is associated with the new text (existing file name)
- Any hard link is associated with the backup file

You can work around this situation in either of the two following ways:

- Copying the data to the backup file, and then writing to the original file.
- Getting the ACL (its best to get the whole property list) using `getproplist()` from the original file and applying it to the new file using `setproplist()`.

Part 2: Administrator’s Guide to Security

Introduction for Administrators 6

The Digital UNIX operating system and its commands, utilities, and subsystems have all been modified to produce the system's trusted computing base (TCB). The changes to the system result in a system that meets the C2 security requirements defined in the *Orange Book*.

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 Digital UNIX 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 Digital 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 Digital 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 trusted system 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 Digital 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 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 Digital 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 Digital 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 Digital 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 track 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 Digital 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*.

6.3.1 Audit Features

Digital 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 from the command line and maintained from the command line or with OSF/Motif based interfaces.

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
  - 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 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. The time before the password expiration warnings is only a default value, and cannot be changed on a per profile 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, Digital 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. Digital'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.

6.4 Windows-Based Administration Utilities

Note

The functions previously performed with the `XIsso` and `XSysAdmin` programs have been moved to other graphical user interfaces (GUIs). The `XIsso` and `XSysAdmin` programs in this release are only interfaces to the other GUIs and support for `XIsso` and `XSysAdmin` will be discontinued after this release.

The following three window-based 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 in the DECwindows environment 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` — You use `dxaudit` to configure the audit system mask and reports, and use `dxaccounts` to set a user's audit mask and controls. Administrators have the flexibility to configure the audit subsystem without the requirement of installing additional C2 security features.

You can find the Audit Manager GUI under the CDE Application Manager→System_Admin→ System_Management_Utilities→Daily_Admin→Audit Manager. If you are using DECwindows environment, start the Audit Manager from the command line with the `/usr/tcb/bin/dxaudit` command.

`dxdevices` — You use the `dxdevices` program to configure devices. In both the CDE and DECwindows environment, the Devices GUI is started from the command line with the `/usr/tcb/bin/dxdevices` 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 (OSFC2SEC400 and OSFXC2SEC400) 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 `secsetup` command, you can select the C2 security features appropriate for your system. The `secsetup` utility is found in CDE under Application Manager→System_Admin→System_Managent_Utilities→Configuration→ Security. The `secsetup` utitlity can also be run from the command line, which is the way it must be run in the DECwindows environment.

The audit subsystem is configurable at kernel link time, regardless of the security level of the system. 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 configured at install time and are enabled and disabled using the `secsetup` utility.

6.5 Administrating the Trusted Operating System

An administrator of a trusted Digital UNIX 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 Digital 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.

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.
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.

On a trusted Digital 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 the ISSO. The trusted Digital UNIX system does not support `sysadmin` and `isso` accounts.

6.5.1.1 Responsibilities of the Information Systems Security Officer

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.)
- Assigns systemwide defaults. Establishes defaults for user login controls and password parameters. (See Chapter 9.)
- 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 (or DECwindows `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 (for example, default command authorizations) 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 (for example, to increase authorizations and privileges). 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.
- 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 (or the DECwindows `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 and command authorization enforcement within a protected subsystem. Administration of these includes assigning subsystem-related authorizations, 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 Digital 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`	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
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 Protected Password Database

The protected password database stores the 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 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 extended profile 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 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

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 pathname can refer to the same physical device. For example, two pathname 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 for details

Setting Up the Trusted System 7

This chapter lists the tasks that must be completed after installation and before the 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 extended authentication mechanism can be set up, the Digital UNIX installation or update must be completed and the optional enhanced security subsets (OSFC2SEC400 and OSFXC2SEC400) 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 " (OSFC2SEC400)

Configuring "C2-Security GUI " (OSFXC2SEC400)
```

The message refers to the installation process, not the security configuration and setup. The `secsetup` script is used to configure or setup the extended authentication mechanism, and the audit and ACL subsystems are kernal options.

7.1.1 Full Installation

A full installation of Digital UNIX (either advanced or basic) brings up the system with no user accounts. If you have DEC OSF/1 Version 1.2 or earlier installed, you need to do a full installation. Run the `secsetup` script before adding accounts.

7.1.2 Update Installation

If you are updating your system from DEC OSF/1 Version 3.2 or higher, all user accounts and databases are preserved, and running the `secsetup` program converts them to the enhanced security format.

If your system was running with the extended authentication mechanism, you should run the `convauth` utility immediately after the update installation. This converts the extended authentication information from the file format to the database format and improves log in performance.

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 1992  /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 `secsetup` asks if you wish to disable segment sharing. The `secsetup` script reports when segment sharing is already disabled.

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 `secsetup` utility is run to configure the extended authentication mechanism. The audit and ACL subsystems are configured as kernel options. (Use the `audit_setup` utility to complete the setup of audit.)

If you are installing Digital UNIX from a console, you will find the audit and ACL subsystems listed as kernel configuration options. They can be selected and built into the kernel during the initial system configuration. (Use the `audit_setup` utility to complete the setup of audit.) Run the `secsetup` script to configure the extended authentication mechanism.

The `audit_setup`(8) reference page and Section 10.2 describe how to set up audit. The `acl`(4) reference page describes the ACL implementation and Section 11.3 describes the Digital UNIX ACL setup.

7.4 The secsetup Command

The `secsetup` command 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, to change the security features, you must reboot your system.

If the you enter a question mark (?) at the prompt, `secsetup` displays information about the choices.

Depending on the security features chosen, when `secsetup` is complete you may need to reboot the system.

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

7.4.1 Setup Questions

Before running `secsetup`, you need to be prepared to answer the following questions:

- Do you want to disable segment sharing?
- Do you want to run `audit_setup` as part of `secsetup`?
- Do you want to edit the configuration file as part of `secsetup`?

7.4.2 Example secsetup Session

The following is an example session showing how security is be setup using `secsetup`. A question mark (?) is entered at points where help is available.

Example 7-1: Using secsetup

```
# /usr/sbin/setld –i | more
OSFC2SEC400   installed   C2-Security (System Administration)
OSFXC2SEC400  installed   C2-Security GUI (System Administration)
# /usr/sbin/secsetup
  All questions asked by this script are for immediate action.
  All changes made have immediate effect unless stated otherwise.
  Please note that changing the current security level leaves
  the system login and password configuration in an inconsistent
  state until the system is rebooted.  Newly-started login
  processes will work as expected, but already-running login
  processes may fail in unexpected ways.  Digital recommends
  backing up the /etc/passwd file before changing the system
  security level and rebooting immediately after the change has
  been made.

Enter system security level(BASE ENHANCED ?)[ENHANCED]: ?


  The default option is to switch the current security level.

  BASE - Discretionary Security Protection:

     Default Digital UNIX style of security features.

  ENHANCED - Controlled Access Protection:
```

Example 7-1: (continued)

```
     A system in this class enforces a more finely grained
     discretionary access control than (BASE) systems.
     Users are individually accountable for their actions
     through login procedures, auditing of security-relevant
     events and resource isolation.

     The audit subsystem can be configured for either of the
     system security levels.


Enter system security level(BASE ENHANCED ?)[ENHANCED]:  Return
ENHANCED security level will take full effect on the next system
reboot.

root
uucpa
nobody
nobodyV
wnn
utest1
utest2
Do you want to create local extended profiles for NIS \
                                        users(yes no ?)[no]: ?


  Creating local extended profiles for NIS users will speed up
  logins.  However, it does not keep network-wide passwords for
  machines running ENHANCED security unless they use NFS to
  share the /tcb/files and /var/tcb/files areas.  An alternative
  is to use the '-s' option to ypbind and share the extended
  profiles with NIS as well as the traditional (BASE) account
  information.

Do you want to create local extended profiles for NIS \
                                  users(yes no ?)[no]:  Return
Successful SIA initialization
Do you want to change the root password now(yes no ?)[yes]: ?


  Changing the root password now will ensure that root logins
  are still possible after rebooting the system.

Do you want to change root password now(yes no ?)[yes]:  Return
Last successful password change for root: UNKNOWN
Last unsuccessful password change for root: NEVER

New password:
Re-enter new password:

Do you wish to disable segment sharing(yes no ?)[no]: ?
```

Example 7-1: (continued)

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

  -rw------- 2 joe  staff  100000 Sep 18 1992  /usr/shlib/foo.so

  When the shared library is used in a program, the
  text part of foo.so may in fact be visible to other
  running processes even though they are not running as
  user joe.  Note that only the text part of the
  library, not the data segment, is shared in this way.
  To prevent this unwanted sharing, any libraries that
  need to be protected can be linked with the -T and -D
  options to put the data section in the same 8-
  megabyte segment as the text section.  The following
  link options should work for any such library:

  % ld -shared -o libfoo.so -T 30000000000 \
                                  -D 30000400000 object_files...

  In fact, this segment sharing can occur with any file
  that is mmap'ed without the PROT_WRITE option as long
  as the mapped address falls in the same memory
  segment as other mmap'ed files.  Any program that
  uses mmap() to examine files that may be highly
  protected can ensure that no segment sharing takes
  place by introducing a writable page into the
  segment before or during the mmap().  The easiest way
  to accomplish this is to mmap() the file with
  PROT_WRITE enabled in the protection, and then use
  mprotect() to make the mapped memory read only.

  Alternatively, to disable all segmentation and avoid
  any unauthorized sharing, answer 'yes' to the question.

Do you wish to disable segment sharing(yes no ?)[no]: yes
Updating configuration file to prevent segmentation...
Configuration file '/etc/sysconfigtab' updated.
Segment sharing will be disabled when the system is rebooted.

Do you wish to run the audit setup utility at this \
                                  time(yes no ?)[no]: ?


  The audit subsystem works with BASE or ENHANCED security,
  recording whatever information is available. There is no audit
  re-configuration between security levels.

Do you wish to run the audit setup utility at this \
                               time(yes no ?)[no]:  Return
```

Example 7-1: (continued)

```
Press return to continue: Return
# shutdown –r now
```

7.5 Configuring Enhanced Security Features

You can configure the enhanced security features individually or you can choose to enable all the enhanced 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 `audit_setup` script to configure audit. See the `audit_setup`(8) and `doconfig`(8) reference pages and Section 10.2 for more information.

7.5.2 Configuring ACLs

You can run the ACL subsystem without installing the security subsets. Configure the ACL subsystem kernel option and reboot your system. See the `doconfig`(8) reference page and Section 11.3 for more information.

7.5.3 Configuring Extended Authentication with NIS

Running the `secsetup` command creates an extended authentication profile database 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), `secsetup` asks if you want to create an extended authentication profile for each user in the NIS server password database. Subsequent changes in NIS passwords are not propagated to the database. The extended passwords now on the local machine are expired and the users must enter a new password the next time they log in.

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

You can use the `edauth` utility to view a specified database (or file if the database does not exist).

7.5.4 Password and Authentication Features Configuration

Enhanced security provides an extended profile to the Digital UNIX authentication mechanism. The following sections explain how you can configure some of the extended profile features for your site's needs. You can configure and remove some authentication and extended password features on your system by customizing the `default` database using the `edauth` program.

Removing or changing features in the `default` database, removes them for all users who are assigned the default settings. The features can be added back or changed on a per user basis by editing the `prpasswd` database.

See the `default`(4), `prpasswd`(4), `ttys`(4), and `edauth`(8) reference pages for more information.

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_minlen` and `u_maxlen` to appropriate values for the site.

An example entry could be as follows:

```
:u_exp#0:u_life#0:u_minlen#5:u_maxlen#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:\
```

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 (adding an at sign (@) at the end negates the action):

```
: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.)

7.5.4.4 Maximum Login Attempts

If break-in evasion is not needed on a per-user basis, you can disable the feature by setting `u_maxtries` to 0. The maximum number of consecutive failed attempts comparison to the number of failures is disabled. The `default` database entry would be as follows:

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

7.5.4.5 Time Between Login Attempts

If the default evasion time (86400 seconds) is not appropriate for your site, change the `u_unlock` field to the right value (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 log in attempts to infinity (no automatic reenabling occurs). For `t_maxtries`, 0 is infinite.

7.5.4.6 Terminal Break-In

If per-terminal break-in evasion is not needed at you site, or the evasion time interval is wrong, modify `t_maxtries` (0 is infinite) or `t_unlock`. For both `u_unlock` and `t_unlock`, 0 is infinite (no automatic reenabling occurs).

7.5.4.7 Time Between Logins

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

The system default log in 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.

7.5.4.8 Per-Terminal Login Records

If you do not want to record per-terminal log in 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 break-in evasion.

7.5.4.9 Automatic Extended Profile Creation

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

7.5.4.10 Vouching

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

7.5.4.11 Encryption

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

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

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

7.6 System Administrator Tasks

On a Digital 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 (or DECwindows `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 Digital 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 (or the DECwindows `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 log in 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 log in. Depending on the procedures established at your site, you may want to unlock all accounts now or unlock them when the users are ready to log in 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.
- Adds each device.

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

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.
- Specifies alternate directories for audit records collected when the system is in multiuser mode.

See Chapter 10 for more information.

7.8 Backing the System Up

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.

Creating and Modifying Secure Devices 8

The 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 Digital UNIX system root access is required to assign devices. The trusted Digital UNIX system currently supports terminals as part of the TCB. This chapter describes how to define those devices to 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
```

Creating and Maintaining Accounts 9

Accounts are created and maintained on a trusted system using the Account Manager (`dxaccounts`) program. The traditional information systems security officer (ISSO), system administrator, and operator roles may be filled by the same person at a given site. An administrator logged in as root can perform the functions of both the security officer (ISSO) and the system administrator that are to set up, modify, and maintain accounts and administer the security aspects of the system. This chapter describes how the `dxaccounts` program are used to create and maintain accounts.

9.1 Using dxaccounts to Perform System Administration Functions

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 template. On a trusted Digital UNIX system, the `dxaccounts` program is used.

9.1.1 Creating User Accounts

Use the Create User Accounts dialog box to create one or more user accounts. To create many accounts in a single session, fill in the information for a new user, click on Apply to save the account, then fill in the information for another user.

The new accounts are created without passwords. Use the `passwd` command to set a temporary passwords for the users.

New accounts are created in a locked state. Users receive an "Account Disabled" message and cannot log in. Use the `XIsso` Modify User Accounts dialog box to unlock the accounts.

9.1.2 Retiring Accounts

Use the Retire User Account dialog box to permanently lock a user account.

9.1.3 Creating Groups

Use the Create New Group dialog box to add a new group.

9.1.4 Modifying the Account Template

An account template is used to establish the account parameters when a new account is created. The ISSO sets default parameters for the system. These defaults apply to all users in the system, but the ISSO can override them for an individual user. Similarly, the system administrator can override these defaults for an individual ISSO.

9.1.5 Modifying User Accounts

The ISSO is responsible for modifying all (non-ISSO) accounts once they have been created by the system administrator. Use the Accounts dialog box to access the Modify User Accounts dialog box.

9.1.6 Modifying the Account Template

An account template is used to establish the account parameters when a new account is created. The ISSO sets default parameters for the system. These defaults apply to all users in the system, but the ISSO can override them for an individual user. Use the Accounts dialog box to access the Modify Account Template box, which provides the screen to make the changes.

9.2 Authentication Subsystem

The authentication subsystem verifys 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 ISSO 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:

- 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))

9.3 Using NIS to Centralize Account Management

You can use the Network Information Service (NIS) to centralize the management of the password, group, and extended profile information. A NIS master server can serve an environment of NIS clients that includes ULTRIX and Digital UNIX machines with and without extended profiles, and non-Digital machines with ordinary UNIX passwords and groups. NIS is documented in the *Network Administration* manual.

9.3.1 Overview of Enhanced Security and NIS User Account Databases

The following section review the account databases and their relationships.

9.3.1.1 BASE Local User Account Database

BASE (BSD) security is the traditional level of security that is available on UNIX systems. The local BASE user account databases are held in `/etc` on each system.

The `/etc/passwd` file in conjunction with the `/etc/group` file comprises the BASE user account database and is used to allow or deny a user access to the system and files on the system in addition to a variety of other functions.

Each line of `/etc/passwd` contains information about one user account. This file contains the users name, UID, password, and several other pieces of information. The `passwd` command is used to change a user's local BASE password.

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

The `adduser`, `addgroup`, `useradd`, `userdel`, `usermod`, and `vipw` commands are used by the system manager to add and change user account information.

9.3.1.2 NIS-Distributed BASE User Account Database

NIS can be used to distribute all or part of the BASE user account database to systems across the network. When you are running NIS over 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 distributed by `ndbm` maps generated from the `/var/yp/src/passwd` and `/var/yp/src/group` files on the NIS server.

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 prepended "+", both databases are read with the information from the `/etc/passwd` file (except for the UID and gid fields) overlaying the information from the NIS distributed user account database.

The `/etc/passwd` file on each system must contain a "+:" as the last entry to allow users from the NIS distributed BASE user account database to log in.

Table 9-1: NIS passwd File 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 is used to change the password in the local BASE user account database, while the `yppasswd` command is used to change the password (and possibly other fields) in the NIS distributed BSD user account database.

To change NIS-distributed BASE user account database entries, the system manager does the following:

1. Edit the `/var/yp/src/passwd` file on the NIS master server.
2. Change directory to `/var/yp`.
3. Run `make` to build and push the updated NIS distributed database.

9.3.1.3 Enhanced Security Local Password Database

Installing the optional enhanced security subsets provides more security features for user accounts. When you are running enhanced security, you have TWO local user account databases.

The local BASE user account database (`/etc/passwd`) remains intact except for the encrypted password, which is held in the enhanced security local protected password database (`/tcb/files/auth.db` or `/var/tcb/files/auth.db`). If the `/etc/passwd` file contains an encrypted password it is ignored unless C1Crypt is being used. The other fields (gecos, shell, and so forth) are used as they normally would be.

The enhanced local protected password database contains the username and UID (to uniquely identify a user), the encrypted password, and a set of keyword-identified values used by enhanced security (see `prpasswd`(4) for a description of these values). Each user account in the local protected password database is backed by the system template (`/etc/auth/system/default`). If a value is not defined for a user in the protected password database, it is taken from the template. There should be entries in BOTH databases (with username and UID both matching) for each user.

A user runs the `passwd` command is to change his protected password.

The CDE Account Manager (or `dxaccounts` program) is used by the system manager to add and change user account information (including passwords).

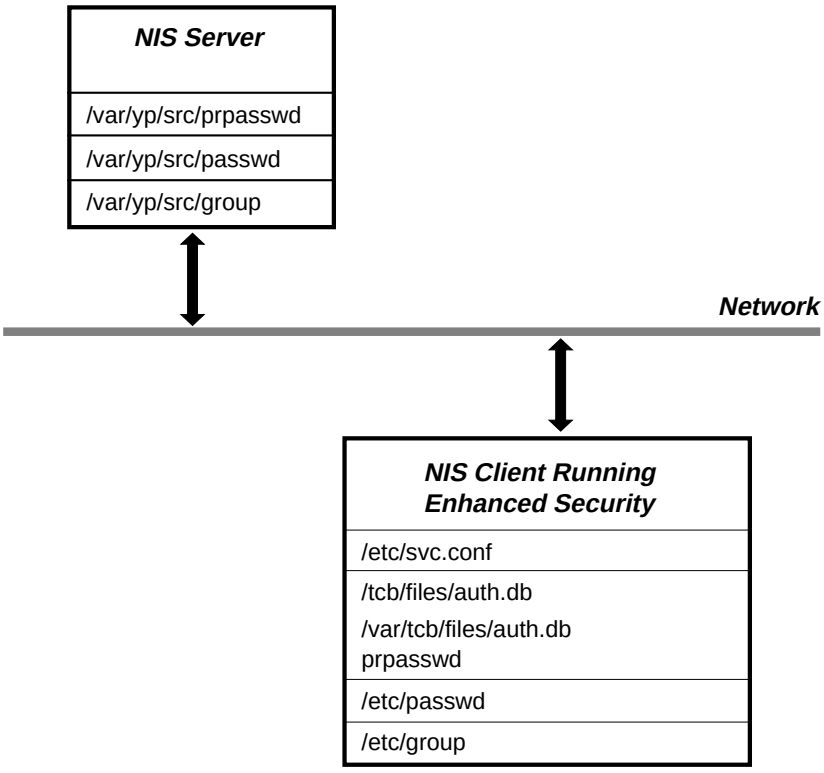
9.3.1.4 NIS and Enhanced Security Database Interaction

NIS can be used to distribute part or all of the enhanced protected password 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.
- The local enhanced security protected password database.
- The NIS-distributed enhanced security password database distributed by `ndbm` maps generated from the `/var/yp/src/prpasswd` file on the master server.

Figure 9-1: NIS and Enhanced Security



The `svc.conf` file "auth=" entry indicates which order the enhanced protected password databases should be searched in (local first or NIS ("yp") first).

The "+" override feature still works the same as always for the BASE user account database (`/etc/passwd`).

Note

When you have NIS running and you are first updating to enhanced security, `secsetup` creates extended passwords for the `+username` entries in the `/etc/passwd` file only if you answer yes to the "Create entries for NIS users" question in `secsetup`.

There is no override feature for the protected password database. An entry in the protected password database is contained completely in either the local

enhanced protected password database or in the NIS distributed enhanced protected password database. Both databases are backed by the local template.

Use the `passwd` command to change the password in the local or NIS distributed enhanced protected password database. Use the `yppasswd` command to change the fields in the NIS-distributed BASE user account database that you've always been able to change with `yppasswd`.

9.3.2 Implementation Notes

To change your password when you are running both enhanced and NIS you should use `passwd` to change the password regardless of whether your protected password database entry is local or distributed. Using the search list in the `svc.conf` file for "auth=", the `passwd` command updates the password in the first protected password database entry it finds for the specified user, even if that entry is in the NIS-distributed protected password database.

The procedure in Section 9.3.6 describes one method that can be used to create an NIS distributed protected password database using the existing local BASE user account database.

It is very important that each protected password database entry exists in only one database either the local protected password database or the NIS-distributed protected password database. The routines that check and manipulate the protected password database information work on the first copy found (as defined in `svc.conf` file). YP routines work on the NIS-distributed protected password database only. This can cause some 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 the root account information. This allows you to still log in to the client systems using the root account if your NIS server is down.

9.3.3 Setting Up a NIS Master Server

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

9.3.3.1 Manual Procedure for Small Databases

The following setup information is specific to an NIS master server that is supporting clients using enhanced security:

1. Ensure that Digital UNIX Version 4.0 or higher is installed.

2. Install the security subsets and setup security. See Chapter 7 for details.
3. Run the `nissetup` program.
   a. When the `nissetup` program first prompts for security (`-s` option to `ypbind`), choose `y` to run `ypbind -s`, which specifies a secure socket.
   b. When the `nissetup` program again prompts for security (`-S` option to `ypbind`), choose `y` and specify a domain name and up to four authorized slave servers.
4. Setup the account Maps using the CDE Account Manager (`dxaccounts`) See Section 9.3.6 for an alternative method of setting up accounts.
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 or restart NIS using the `/sbin/init.d/nis start` command.

9.3.3.2 Automated Procedure for Large Databases

If you have a large existing NIS distributed BASE user accounts database you can automate the creation of the NIS distributed protected password database by doing the following:

1. Ensure that Digital UNIX Version 4.0 or higher is installed.
2. Install the security subsets and setup security. See Chapter 7 for details.
3. Run the `nissetup` program.
   a. When the `nissetup` program first prompts for security (`-s` option to `ypbind`), choose `y` to run `ypbind -s`, which specifies a secure socket.
   b. When the `nissetup` program again prompts for security (`-S` option to `ypbind`), choose `y` and specify a domain name and up to four authorized slave servers.
4. Enter the following command:

   ```
   # convuser -Mc
   ```

9.3.4 Setting Up a NIS Slave Server

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 an NIS slave server that is supporting clients using enhanced security:

1. Ensure that Digital UNIX Version 4.0 or higher is installed.
2. Install the security subsets and setup security. See Chapter 7 for details.
3. Run the `nissetup` program.
   a. When the `nissetup` program first prompts for security (`-s` option to `ypbind`), choose `y` to run `ypbind -s`, which specifies a secure socket.
   b. When the `nissetup` program again prompts for security (`-S` option to `ypbind`), choose `y` and specify a domain name and up to four authorized slave servers.
4. Edit the `/etc/svc.conf` file to include a yp entry for `auth`. The entry should be as follows: `auth=local,yp`.
5. Start or restart NIS using the `/sbin/init.d/nis start` command.

9.3.5 Setting Up a NIS Client

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

1. Ensure that Digital UNIX Version 4.0 or higher is installed.
2. Install the security subsets and setup security. See Chapter 7 for details.
3. Run the `nissetup` program.
   a. When the `nissetup` program first prompts for security (`-s` option to `ypbind`), choose `y` to run `ypbind -s`, which specifies a secure socket.
   b. When the `nissetup` program again prompts for security (`-S` option to `ypbind`), choose `y` and specify a domain name and up to four authorized slave servers.
4. Install enhanced security and run the `secsetup` program.
5. Edit the `/etc/svc.conf` file to include a NIS (yp) entry for `auth`. The entry should be as follows: `auth=local,yp`.
6. Start or restart NIS using the `/sbin/init.d/nis start` command.

9.3.6 Moving Local Accounts to NIS

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

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

9.3.7 Backing Out NIS

If you need to remove the NIS support from your trusted client system first disable NIS for authentication data, then enter the following command on the client:

```
# edauth -gN | edauth -sLC
```

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

Administering the Audit Subsystem 10

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

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.

It is important that you inform users of the purpose and, in general terms, the nature of the auditing performed on the system. Represent 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.

10.1.1 Files Used for Auditing

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.`*nnn*	Default log file for the audit subsystem when `audit_setup` is used.
`/etc/sec/event_aliases`	A set of aliases to represent the set of events which can be audited.
`/etc/sec/auditd_clients`	A list of the remote hosts that can send remote audit data to the local audit log.
`/etc/sec/audit_events`	A list of the events which can be used as input for `auditmask`.
`/etc/sec/site_events`	A file designated for site-defined events.
`/etc/sec/auditd_cons`	Default log file for console audit messages.
`/etc/sec/auditd_loc`	A list of alternate paths and hosts for audit logs.

10.1.2 Auditing Tools

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

- The audit subsystem, which has powerful features unique to the trusted Digital UNIX operating system. This is a better way to perform security-relevant auditing.
- Traditional UNIX operating system logging features, such as the system accounting files and the `last` command.

The Digital UNIX audit subsystem provides a choice of the events to be logged, flexible data reduction of the audit log, and ease of maintenance. The following commands are used with the audit subsystem:

`audit_setup`	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 and administers audit data storage.

`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. They are discussed in greater detail later in this chapter.

The Digital UNIX audit subsystem records activities in any file at any location chosen by the system administrator. The default log file is `/var/audit/auditlog.`*nnn*, where *nnn* is the generation number of the file, a number between 000 and 999.

The subsystem is capable of recording a wide range of events. You can choose a list of events to log for all users and then add or remove events from that list on a per-user basis to tailor the logging to individual users. The `audit_tool` program enables you to filter the audit log file and produce reports focusing on the information you need.

10.2 Setting Up the Audit Subsystem

The following sections explain how to setup a basic audit subsystem.

10.2.1 Set Up Questions

Before setting up the audit subsystem, you need to make the following decisions about how it is to run on your system:

- Determine the location for your local audit data. (Section 10.4.1)
- Determine the action to be taken when the local audit log overflows. (Section 10.4.1.1)
- Determine if there is a need to store local audit data remotely or to receive remote audit data. (Section 10.7)
- Decide on a list of events and system calls to be logged. You might have more than one list if you are logging different events at different times. (Section 10.10)
- If you are going to require per-user audit profile setup or the `dxaudit` program, ensure that you have the optional enhanced security subsets (OSFC2SEC4xx for per-user audit profile and OSFXC2SEC4xx for

dxaudit) installed. You can check as follows:

```
$ ls -l /usr/.smdb./OSF*C2SEC4??.lk
-rw-r--r--  1 root  system  0 Nov  8 11:02 \
                              /usr/.smdb./OSFC2SEC400.lk
-rw-r--r--  1 root  system  0 Nov  8 11:08  \
                              /usr/.smdb./OSFXC2SEC400.lk
```

The presence of the lock files (OSFC2SEC400.lk and OSFXC2SEC400.lk) indicates that the enhanced security subset is installed on your system.

- Decide if you are going to setup audit using the audit_setup script or manually setup audit using the audit utilities. Using the audit_setup script is the recommended method.

10.2.2 Using the audit_setup Script

The audit_setup script configures your default audit subsystem. It sets parameters for auditd and the auditmask, which are used by one of the rc3 scripts to start audit whenever your system goes to multiuser (run level 3). For details, see audit_setup(8).

If you use the audit_setup script, enter the following from the command line:

```
# /usr/sbin/audit_setup
```

Example 10-1 shows a sample audit_setup session:

Example 10-1: Using the audit_setup Script

```
*****************************************************************

           Audit Subsystem Setup Script

*****************************************************************

  The following steps will be taken to set up audit:
    1) establish startup flags for the audit daemon,
    2) establish startup flags for the auditmask,
    3) create the /dev/audit device (if needed),
    4) configure a new kernel (if needed).

  Do you wish to have security auditing enabled as part of
  system initialization (answer 'n' to disable) ([y]/n)?  y

    ----------------------------
     Audit Daemon Startup Flags
    ----------------------------

  Some of the options to 'auditd' control:
    1) destination of audit data,
    2) destination of auditd messages,
    3) action to take on an overflow condition,
```

Example 10-1: (continued)

```
    4) enable accepting audit data from remote auditd's.


  Destination of audit data (file|host:) [/var/audit/auditlog]? Return
  Directory /var/audit/ does not exist; create it now (y/[n])?  y

  Destination of auditd messages [/var/audit/auditd_cons]?   Return

  Action to take on an overflow condition may be one of:
    1)  change audit data location according to '/etc/sec/auditd_loc'
    2)  suspend auditing until space becomes available
    3)  overwrite the current auditlog
    4)  terminate auditing
    5)  halt the system

  Action (1-5) [1]?   Return

    List in '/etc/sec/auditd_loc' the alternate directories in
    which to store audit data, and host names to which to send data.

    Do you wish to edit /etc/sec/auditd_loc now (y/[n])?   Return

  Accept data from remote auditd's (y/[n])?  y

    Don't forget to place names of remote hosts from which data
    may be accepted into '/etc/sec/auditd_clients'.

    Do you wish to edit /etc/sec/auditd_clients now (y/[n])?  y

?auditd_clients
a
alpha1
alpha1.sales.dec.com
.
1,$n
1          alpha1
2          alpha1.sales.dec.com
w
q

  Further options are available for advanced users of the audit system
  (please refer to the auditd manpage).  If you wish to specify any
  further options you may do so now (<Return> for none):   Return


  Startup flags for 'auditd' set to:
    -l /var/audit/auditlog -c y -o changeloc -r -s

  Is this correct ([y]/n)?  y

    -------------------------
     Auditmask Startup Flags
    -------------------------

  The auditmask establishes which events get audited.  This can be
  specified by:
    1) having the auditmask read a list of events from a file,
      -or-
    2) specifying a list of events on the command line.
```

Example 10-1: (continued)

```
  Events can refer to syscalls, trusted events, site-defined events, or
  alias names.


  The file '/etc/sec/audit_events' contains a list of all auditable
  system calls and trusted (application) events.  You may either
  modify this file or use it as a template.

  The file '/etc/sec/event_aliases' contains a set of aliases by which
  logically related groupings of events may be constructed.  You may
  modify this set of aliases to suit your site's requirements.


  Please enter the filename containing the event list or enter * to
  indicate that the individual events will be specified on the command
  line, or enter <Return> for no events:  /etc/sec/audit_events

  Do you wish to edit /etc/sec/audit_events now (y/[n])?   Return

  The auditmask also sets various style flags such as:
    1) 'exec_argp'   - audit argument vector to exec system calls
    2) 'exec_envp'   - audit environment vector to exec system calls
    3) 'login_uname' - audit recorded username in failed login events


  Enable exec_argp ([y]/n)?   Return

  Enable exec_envp (y/[n])?   Return

  Enable login_uname ([y]/n)?   Return


  Startup flags for 'auditmask' set to:
     -s exec_argp -s login_uname < /etc/sec/audit_events

  Is this correct ([y]/n)?   Return

    ----------------------
     System Configuration
    ----------------------

  Configuration file name (/sys/conf/ALPHA1)?  Return

  Checking booted kernel '/vmunix' and config file '/sys/conf/ALPHA1'....

  /vmunix on ALPHA1 is already configured for security auditing.

  Would you like to start audit now ([y]/n)?   Return

    '/usr/sbin/auditd' started.
    '/usr/sbin/auditmask' set.

  ***** AUDIT SETUP COMPLETE *****

#
```

10.3 Selecting Audit Events

You can specify the auditing of events as follows:

- With the `auditmask` command, you assign values to the system audit mask, which determines the events that are stored in the audit log for all users.
- With the user audit mask, you can specify logging events for individual users.

The system audit mask and the user audit mask for the particular user determine the events logged for that user. The way the two masks are combined is controlled by the user's audit control flag. The user audit mask and the user audit control flag are part of the user's account and are stored in the protected password database (`/var/tcb/files/auth.db`). If the `authcap` files are not used, an OR operation is performed on the default system and the default user auditmasks (the user mask is empty).

The user audit mask and audit control flag enable you to designate events to be logged for the user in addition to those that are logged for all users, or to selectively exclude the user from the auditing of certain events.

The syntax of the `auditmask` command is as follows:

/usr/sbin/auditmask [*options*] [*event_specification*]

The *event_specification* lets you designate the event to be logged and whether only successful occurrences of the event, only failed occurrences, or both are to be recorded. An *event_specification* has the following form:

event[:success:failure]

In place of *`event`*, enter the name of the event you want audited. In most cases an audit event corresponds to a system call, trusted event, site-defined event or event alias.

In place of *`success`* enter one of the following:

1 To log the event when it succeeds.

0 To suppress logging the event when it succeeds.

In place of *`failure`*, enter one of the following:

1 To log the event when it fails.

0 To suppress logging the event when it fails.

If you do not specify either success or failure logging, then both are logged; that is, `login` has the same effect as `login:1:1`.

You can specify multiple events by listing individual event specifications separated by spaces. For example, to specify failed login attempts, successful

and failed file opens, and successful file closes, you would enter the following:

```
# /usr/sbin/auditmask login:0:1 open close:1:0
```

Once an event has been designated for logging, logging of the event continues until you explicitly turn it off by specifying the following:

*<event>***:0:0**

Use the `-n` option to disable all logging.

See Section 10.10 for more information on audit events. For a complete description of selecting audit events, see the `auditmask`(8) reference page.

10.3.1 Event Aliases

You can create an event alias to group events to be audited. You define the event aliases in the `/etc/sec/event_aliases` file. The format is a series of alias entries, as follows:

alias: event[:x:y] [event[:x:y] ...]

Each event is either a system call, trusted event, site event, or another alias, and the :x:y is the same success/fail notation used in `auditmask` and `audit_tool`. Continuation lines are allowed.

See Appendix B for a sample `/etc/sec/event_aliases` file.

10.3.2 Object Selection and Deselection

Auditing is capable of generating enormous amounts of data. To obtain usable audit logs, mechanisms to specify when audit data is generated are very important. Another pre-selection mechanism offered by Digital UNIX auditing is the file system object selection/deselection mode which works in conjunction with the event selection mode.

Some events, such as `mount` and `reboot`, are operations which affect system state; other events, such as `open` and `stat`, are operations which affect specific files. While all `reboot` attempts might be security relevant, not all file `open` events are (based on the site security model) relevant. The file system object selection and deselection modes provide a further level of granularity for events which operate on files.

The object selection and deselection modes are divided as follows:

Selection — The file selection mode allows administrators to specify a set of files against which data access operations can generate audit data, while those same operations on other files can not generate audit data.

The data access operations are:

```
open        stat
close       lstat
link        dup
lseek       revoke
access      readlink
fstat       dup2
read        getdirentries
```

Using object selection it is possible, for example, to audit the `open`s of the `/etc/passwd` and `/.rhosts` files while not auditing `open` events of `/tmp/xxxx` files.

Deselection

The file deselection mode allows administrators to specify a set of files against which data access operations can not generate audit data, while other files are normally audited. The set of operations is the same as that listed for file selection. File `open`'s for write or truncate access, however, do not get deselected.

The result is that it is now possible, for example, to not audit accesses to `/usr/shlib/libc.so`, but still audit the opening of the `/etc/passwd` file.

Note that processes with an `audcntl` flag of AUDIT_USR do not have their auditing reduced through the selection/deselection mechanism.

Although the system object selection and object deselection states 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.

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 deselect a list of files:

```
# auditmask –s obj_sel
# 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 –X desel_file:0
selection: on  => off  --  /etc/motd
```

```
selection: on  => off  --  /etc/fstab
selection: on  => off  --  /etc/passwd
```

10.3.3 Targeting an Active Processes

You can audit a process in real time by using the `-p` option to `auditmask`. Example 10-2 shows how you might investigate a process started by a user logged in as `guest`.

Example 10-2: 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.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.001 -Bfw    5
USERNAME    PID    RES/(ERR)   EVENT
--------    ---    ---------   -----
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 )
```

Example 10-2: (continued)

```
jdoe      * 23592  0x0         execve ( /sbin/ps ps gax )
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 )
jdoe        23563  0x4         open ( savethis 0x1 )
jdoe        23563  0x4         open ( /tmp/edtmpAA 0x602 100640 )
jdoe        23563  0x4         open ( savethis 0x601 0640 )
jdoe        23563  0x5         open ( /tmp/edtmpAA 0x0 )

^C                          6
--interrupt:  exit (y/[n])?  y
#
```

1 Find out what process the 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 abbreviated audit report. Note that the audit log argument comes from the previous `auditd -w` command.

6 Exit the `audit_tool` program with a Ctrl/C (auditing continues).

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

10.4 Audit Log Files

The audit subsystem uses several log files to collect audit data.

10.4.1 The auditlog File

All of the security-relevant audit data is collected in the `auditlog.`*nnn* file. Because of possible system configuration differences, the default location is different depending on how the audit subsystem is setup. If it is setup using the `audit_setup` script, the default location is at `/var/audit/auditlog.`*nnn*; if audit is setup using the `auditd`, the default location is at `/var/adm/auditlog.`*nnn*. In either case, the administrator can choose another location and file name.

The audit daemon increments the number, *nnn*, by one each time a new audit log is created. Some of the circumstances that cause the creation of a new audit log are as follows:

- A new log is requested (`auditd -x`) either by the system administrator or from `crontab`.
- Auditing is started.
- An overflow condition occurs.

10.4.1.1 Audit Log Overflow

When the file system for the current audit log fills to the specified level, a warning message is displayed at `/dev/console` or the specified audit console, and `auditd` takes its overflow action. You can specify the action to be taken when the audit log overflows by using either the `audit_setup` script or the `auditd -o` command. The actions you can take are as follows:

- Change to the next directory or host as specified in the `/etc/sec/auditd_loc` file.
- Suspend auditing.
- Overwrite the current audit log file. This causes the loss of previously logged audit data.
- Terminate the audit daemon.
- Halt the system.

The `-o changeloc` option to `auditd` allows the administrator to specify a list of directories or hosts to which `auditd` can store data. The `auditd` program on the host machine determines the path. If `auditd` reaches the ''virtual limit,'' which is established with the `-f` option, it continues to output data into the next directory or host as specified in the `/etc/sec/auditd_loc` file. (The directories specified in `/etc/sec/auditd_loc` should be on different disk partitions.) If `auditd` cycles through all the directories listed in the `/etc/sec/auditd_loc` file, it will continue to output data into the last specified directory until the physical limit of the directory is reached.

When `auditd` fails because it has filled the last directory or cannot overwrite the log file, as specified by `auditd -o overflow`, the system shuts down with the following console message:

```
% auditd: overflow condition -> shutdown
```

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

10.4.1.2 Remote Audit Logs

Audit data can be sent to another system or can be received from other systems. The `-l` flag to `auditd` specifies a remote host to receive the audit data. If the remote site stops receiving, the local daemon stores its data locally as specified with the `-o` and `-r` flags to `auditd`.

The `-s` flag to `auditd` allows the audit daemon to accept audit data from other audit daemons whose host names are specified in the file `/etc/sec/auditd_clients`. The `/etc/sec/auditd_clients` file is site-specific and lists one host name per line.

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

10.4.2 Console Messages

Console messages generated by `auditd` are directed to a log file named `auditd_cons`. The default location is `/var/audit/auditd_cons`. You can choose a different location using the `audit_setup` script.

10.4.3 Creating Your Own Log Entries

By using the `audgen` command, you can include a message of your choosing in the audit log. The command builds an audit record consisting of user-supplied text and the audit ID, UID, PID, IP address, and timestamp. For example, if you wanted to annotate the log file, you might want to create an entry like the following:

```
# /usr/sbin/audgen "Begin recording file opens by J. Doe."
```

You must enclose the audit log text in quotation marks or each word in the text will appear on a separate line when the text is later retrieved by the audit tool.

To log the event, `AUDGEN8` must be one of the events designated for auditing with the `auditmask`. To extract records logged with the `audgen` command, include `audgen` in the list of events to be retrieved by the `audit_tool` command.

See the `audgen`(8) reference page.

10.5 Configuring the Audit Subsystem Using auditd

Although using the `audit_setup` script is the recommended way to configure your audit subsystem, you may want to do some tasks manually using the audit subsystem daemon, `auditd`.

10.5.1 Displaying Information About the Audit Subsystem

The `auditd` command provides several ways to obtain information about the audit subsystem:

- To display a brief help menu, use the `-h` option.
- To learn the location of the current audit log file, use the `-q` option.
- To learn the current options specified for the audit daemon, use the `-w` option.

10.5.2 Designating the Location of the Audit Log File

To designate the local pathname to which audit data is written, use the `-l` option to `auditd` followed by the pathname you want for the audit log. The file name should be of the form `auditlog[.`*nnn*], where *nnn* can be a number from 000 to 999, inclusive. The audit daemon appends the number automatically, incrementing the number by one each time a new audit log is generated. If you specify the number, the range is from the specified number to 999. For example, to create a new series of unique log files with the base name `special_log`, enter the following:

```
# auditd -l /var/audit/special_log
```

By specifying a remote host as the argument to the `-l` option, you can send the audit data generated locally to a remote host running the audit daemon. For example, to send the your audit data to a remote host named `peanuts`, enter the following:

```
# auditd -l peanuts:
```

The remote daemon sets the path to the audit log on the remote system.

10.5.3 Designating a Fallback Location for Audit Data

The `auditd -r` command reads a list of local directories or host names into which `auditd` switches its audit log file when an overflow condition is reached. When sending data to a remote host, the directories are determined by the remote host's `auditd`. It is prudent to designate a fallback destination that is in a different file system than the one for the current audit log.

The list of fallback locations is maintained in the `/etc/sec/auditd_loc` file. The `auditd_loc` file is site-specific and lists one host name (in host: format) or one directory path per line. Example 10-3 is an example of what an `/etc/sec/auditd_loc` file might look like.

Example 10-3: Sample /etc/sec/auditd_loc File

```
host1:
host2:
/var/alternate/auditdata1
/var/alternate/auditdata2
/var/alternate/auditdata3
```

The `-r` option is used when the overflow action (`-o` option) is set to `changeloc`. The following example shows how to tell `auditd` the location of your fallback directories:

```
# auditd -o changeloc -r
```

10.5.4 Designating a Destination for Audit Log Status Reports

To provide you with up-to-date information on the status of the audit log, the audit subsystem sends messages about such things as log overflow conditions and the rollover of the current log file to a new file. Use the `auditd -c` command followed by a pathname to specify a device or local file as the destination of those messages. For example, to direct the status reports for your audit logs to a file named `/usr/staff/r0/jdoe/log_status`, enter the following:

```
# auditd -c /usr/staff/r0/jdoe/log_status
```

10.5.5 Protecting Against Audit Log Overflow

The audit subsystem enables you to protect against the possibility of lost audit information due to the log file filling all the available space of a disk partition. The `auditd -o` command followed by an action string lets you specify an action to be taken by the system in the event of an overflow condition. You can choose the following actions:

`changeloc`	Change to the next directory or host listed in the `/etc/sec/auditd_loc` file.
`halt`	Shut down the system.
`kill`	Terminate `auditd` (stops auditing).
`suspend`	Suspend auditing until space is made available.
`overwrite`	Overwrite the current log file.

For example, on an overflow situation, to cause the audit logs to be sent to the next directory or host listed in the `/etc/sec/auditd_loc` file, enter

the following:

```
# auditd –o changeloc
```

10.6 Starting Audit

The first invocation of `/usr/sbin/auditd` spawns the daemon; subsequent invocations detect that an audit daemon is already running and communicate with it, passing the specified option parameters. The first invocation of the daemon also turns on auditing for the system (`audcntl`).

You can start the audit daemon using just the `auditd` command with no options and can then configure the audit subsystem using the `auditd` command for each individual option as described in the following sections. You can also configure and start the audit subsystem from a single command that includes the configuration options. The following is an example of a combined configure and start audit command line:

```
# auditd –o changeloc –c ~jdoe/log_status \
       –r /etc/sec/auditd_loc –l peanuts:/var/audit/special_log
```

10.6.1 Turning Off Audit

The `auditd –k` command stops the audit daemon, thus disabling the audit subsystem. For example, to immediately stop auditing, enter the following:

```
# auditd –k
```

Avoid using this option during normal system operation.

10.6.2 Starting a New Audit Log

To start (or roll over) a new auditlog with the number of the current auditlog incremented by 1, use the `–x` option to `auditd`. When you use this option, the old audit log is automatically compressed by the `compress` command. You can do this while the audit subsystem is active.

The name of an audit log file is always appended with a generation number, which ranges from 000 to 999. If the current audit log is `auditlog.275`, the `–x` option creates a new file, `auditlog.276`, and begins writing audit data to it.

10.7 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:
   ```

4. From the audit hub you can designate options for an audit daemon serving a remote host by using the `auditd` command as usual to specify the options. On this hub, you will have a dedicated `auditd` for each remote connection. Include the `–p` option to designate the ID of the remote audit daemon to receive the options. To learn the ID of an audit daemon serving a remote connection, enter the following commands:

   ```
   # /usr/sbin/auditd –w
   MASTER AUDIT DAEMON SERVER:

   Audit data and msgs:
     -l) audit data destination       = /var/audit/auditlog.039
     -c) audit console messages       = /var/audit/auditd_cons

   Network:
     -s) network audit server status (toggle)   = on
     -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 DAEMON #1 SERVING vijy.research.dec.com:

   Audit data and msgs:
     -l) audit data destination = \
                    /var/audit/auditlog:vijy.research.dec.com.040
     -c) audit console messages       = /var/audit/auditd_cons

   Network:
     -t) connection timeout value (sec)   = 4

   Overflow control:
     -f) % free space before overflow condition = 10
     -o) action to take on overflow  = overwrite current auditlog

   # /usr/sbin/auditd –p 1 –l /var/audit/vijy –dq
   ```

```
/var/audit/vijy.041
```

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.

10.8 Processing Audit Log Data

The `audit_tool` command enables you to process and filter data stored in the audit log and to display the audit information in a format you can read. The command handles both compressed and uncompressed files.

The `audit_tool` command has the following syntax:

/usr/sbin/audit_tool [*options*] *filename*

Use *`filename`* to designate the audit log from which audit information is to be extracted.

Within a single option type, audit records are returned for each use of the option. For example, assuming the name of the audit log is `auditlog.100`, you can retrieve the records of all `open`, `close`, and `rename` events using the `–e` option as follows:

```
# /usr/sbin/audit_tool –e open –e close –e rename /var/adm/auditlog.100
```

When you mix different options, only audit records that match the specified attributes are returned. For example, to get reports on only `open` events that are also associated with user name `smith`, you can use the following command:

```
# /usr/sbin/audit_tool –e open –U smith /var/adm/auditlog.100
```

The `audit_tool` command can be highly selective in the audit records it extracts. For example, the following command extracts only records of `open`, `close`, or `rename` events that are associated with user name smith

or brown.

```
# /usr/sbin/audit_tool –e open –e close –e rename –U smith \
                                                  -U brown auditlog.100
```

When specified without an argument, the `audit_tool` command displays a brief help message.

The following sections describe some often-used features of `audit_tool`. For a complete description, see the `audit_tool`(8) reference page.

10.8.1 Using audit_tool Interactively

To run `audit_tool` interactively, use the `–i` option, which displays each option, along with its current or default setting. Enter your choice, or press Return to accept the current setting or default.

When in interactive mode, the default is to retrieve all audit records.

10.8.2 Selecting Audit Records

You can select records that are associated with user identity in four ways:

- Select records by user name.

  Use the `–U` option to specify one or more user names. Records with user names that match your input are then returned. Note that a user name is associated with a log record only if the event `login` is audited (see Section 10.10.1). The default is all user names.

- Select records by effective UID.

  Use the `–u` option to specify one or more UIDs. Log entries for processes with effective UIDs that match your input are then returned. The default is all UIDs.

- Select records by real UID (RUID).

  Use the `–r` option to specify one or more RUIDs. Log entries for processes with RUIDs that match your input are then returned. The default is all RUIDs.

- Select records by audit ID (AUID).

  Use the `–a` option to specify one or more audit IDs. Log entries for processes with audit IDs that match your input are then returned.

10.8.3 Generating a Report for Each Audit ID

Use the `–R` option to `audit_tool` to generate an ASCII report for each audit ID associated with the events being logged. All events with a common audit ID are placed in a report with the name `report.`*n*, where *n*, is the

actual audit ID. You can specify a file name prefix by using an argument to the `-R` option, such as `-R foo_`.

10.8.4 Selecting Audit Records Within a Time Range

Use the `-t` option to `audit_tool` followed by a time specification to designate a start time. Then only those records with a timestamp equal to or greater than that start time are selected. Use the `-T` option followed by a time specification to designate an end time. Then only records with timestamps equal to or less than that end time are selected. Use `-t` and `-T` together to limit the audit records retrieved to only those that occurred within a given period.

The format for start and end times is yymmdd[hh[mm[ss]]]. You can specify only one start time and one end time. The default is to select for all records.

10.8.5 Selecting Audit Records for Specific Events

Use the `-e` option to `audit_tool` to retrieve audit records that match a list of designated events and their success and failure status.

The event specification has the following syntax:

event[:*success*:*failure*]

In place of *event*, enter the name of the event or an event alias for which you want to retrieve information. The default list of all auditable events is in the `/etc/sec/audit_events` file (see Appendix B).

The event selection for record retrieval is the same as event selection of the `auditmask` command. In place of *success* enter the following:

1 To retrieve successful instances of the event.

0 To retrieve failed instances of the event.

In place of *failure*, enter the following:

1 To retrieve successful instances of the event.

0 To retrieve failed instances of the event.

If you do not specify success and failure extraction, both are included in the audit report.

For example, the following command retrieves audit information only for

failed attempts to change file ownership:

```
# audit_tool -e chown:0:1 filename
```

10.8.6 Performing Continuous Audit Reporting

Use the `-f` option (in conjunction with the `-d` option to `auditd`) to have the `audit_tool` program read the audit log continuously, generating a report as it goes. The log is read even after the end of the file is reached. This allows you to extract audit data as it is being written to the audit log, which gives you the current audit information as it is generated.

10.8.7 Selecting Audit Records for Process IDs

To retrieve records associated with specific process IDs (PIDs), use the `-p` option and enter the PIDs you are interested in. The default is to select for all process IDs. If the specified PID is negative, the absolute value of the PID is selected as well as any of the PID's descendants.

10.8.8 Filtering Out Specific Audit Records

You can create a deselection file to filter out audit records that you do not want to see. In this way, you minimize the number of records to review.

Create the file at any location and with your choice of a file name. Edit this file to add a line for each event to be filtered out. The lines have the following syntax:

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 use the `audit_tool` command. To filter audit data using a deselection file, include it on the

audit_tool command line with the -d option as follows:

```
# audit_tool ... -d filterfile4
```

10.8.9 Processing ULTRIX Audit Data

The audit_tool.ultrix program displays audit reports on a Digital UNIX system from audit data collected on ULTRIX systems. With the exception of the -g and -G options (equivalent to the -v and -V options for audit_tool), audit_tool.ultrix is the same as audit_tool. See the audit_tool(8) reference page and Section 10.8 for more information.

10.9 Site-Defined Audit Events

The Digital UNIX audit subsytem allows sites to define their own audit events (referred to as site-defined events). This is useful for applications that want to generate records specific to their requirements.

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

10.9.1 System Administrator's Responsibilities

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 range for the site event numbers is between MIN_SITE_EVENT (defined in the <sys/audit.h> file) and INT_MAX (defined in the <machine/machlimits.h> file).

The site_events file contains one entry for each site-event. Each site-event entry may contain any number of subevents. Both preselection (see auditmask(8)) and postreduction (see audit_tool(8)) capabilities are supported for site events. Postreduction capabilities are also supported for subevents.

The syntax for the /etc/sec/site_events entries is as follows:

event_name event_number [, subevent_name subevent_number ...] ;

The following is an example of a /etc/sec/site_events file:

```
essence 2048,
    ess_read 0,
    ess_write 1;
rdb 2049,
    rdb_open 0,
    rdb_close 1,
    rdb_read 2,
    rdb_write 3;
```

```
decinspect 2050;
```

In the first entry, `essence` is the event, 2048 is the event number, `ess_read` is the first subevent, 0 is the first subevent number, `ess_write` is the second subevent, and 1 is the second subevent number.

10.9.2 Trusted Application Responsibility

The trusted application generates audit data using the `audgenl` routine. This routine gives the programmer control over what data appears in the audit log.

The following code fragment generates audit data from a `rdb_close` in a trusted application:

```
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\n" );

/* generate audit data */
else if (audgenl ( event_num, T_SUBEVENT, subevent_num, T_CHARP,
                                "Trusted RDB V1.0", 0 ) == -1)
     perror ( "audgenl" );
```

It is recommended, although not required, that you include a `T_CHARP` and "*event name*" argument pair to `audgenl()`. This makes analyzing the audit data with `audit_tool` easier on a system that does not have the `site_events` file.

Example 10-4 is the audit record generated from the example code:

Example 10-4: Layered Product Audit Record

```
audit_id:    0             ruid/euid:   0/0      (username: root)
pid:         2394          ppid: 2265            cttydev: (6,0)
event:       rdb
subevent:    rdb_close
char param:  Trusted RDB V1.0
ip address:  16.153.127.241 (alpha1.sales.dec.com)
timestamp:   Tue Mar 29 15:48:13.65 1994 EST
```

See Chapter 19, `aud_sitevent`(3), and `audgenl`(3) for more information.

10.9.3 Managing Your Own Audit Data

If you want to create your own audit log, use the `audgen()` system call in place of `audgenl()`. If the *`size`* argument given to `audgen()` is nonzero, the audit data is not passed into the system audit data stream, but is copied out to the *`userbuff`* specified in `audgen()`. The trusted application can then send the data in *`userbuff`* to a unique log file. You can read your audit log using the `audit_tool` utility. See the `audit_tool`(8) reference page for more information.

When `audgen()` is called, the system provides the following audit data:

- AUID, RUID, EUID
- PID, PPID
- Device
- Timestamp
- User name
- IP address

The application provides the balance of the audit data as specified in the arguments to `audgen()`. See the `audgen`(2) reference page for more information.

10.9.4 Changing the Site Event Mask

The site event mask is the kernel representation of the contents of the `/etc/sec/site_events`, just as the system syscall audit mask is the kernel representation of whatever file is passed into the `auditmask` command. Unlike the system call audit mask, which is of known fixed size, the size of the site event mask is determined by the customer. The size of the site event mask can be changed using the `sysconfig` command. The details, such as the # events per byte, default, minimum and maximum size are found in the *System Tuning and Performance Management.*

10.10 Suggested Audit Events

When deciding which system events to audit, you need to keep in mind that auditing uses system resources. Logging a large number of events to allow for in-depth auditing has a cost in terms of system performance.

The safest practice is to log all events at all times. But unless your system requires very thorough security protection, this is unnecessary. Typically, you can achieve an adequate level of protection by regularly logging a limited number of events and by performing deeper logging at more widely spaced intervals. This provides a reasonable level of auditing capability and minimizes the impact on system performance.

Note

If you vary the depth of logging, avoid signaling to the user community the times when the deep audits occur; otherwise, would-be violators, hoping to avoid detection, will avoid those times for their illicit activity.

You should audit the `logout()` event. This makes the `audit_tool` program run faster during postreduction.

10.10.1 Dependencies Among Audit Events

Some information in the audit log represents information based on previous audited events. For example, the LOGIN event associates a login name with an RUID. Subsequent occurrences of that RUID (for a given process) can then be associated with a login name. This 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 1993
```

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 1993
```

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 1993
```

Other examples of state-dependent information include the mapping of file descriptors to pathnames, current directory, and process name.

If you are auditing events that use file descriptors (for example, `close` or `ioctl`) and you want to see the associated pathname in the audit record, it is necessary to have been auditing the event that associated the pathname with the file descriptor (for example, `open`, `socket`, or `bind`), as well as other events that manipulate the file descriptors (`dup`, `dup2`, or `fcntl`).

To see the current directory, it is necessary to audit the `chdir` and `chroot` events.

To see the process name, it is necessary to audit the `exec` event (`execve`, `execv`, or `exec_with_loader`).

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()`, but you should then use the `-F` option to the `audit_tool` program.

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

10.10.2 Auditable Events

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

- System calls
- Trusted events
- Site-defined events

The events listed in the `/etc/sec/audit_events` file are the system calls that can be audited. See Appendix B for the default list of the system calls.

A trusted event is an event that is associated with a security protection mechanism; it does not always correspond directly to a system call. The AUDGEN8 and LOGIN trusted events correspond to the use of the audgen and login commands. A list of the other trusted events that relate to auditing and authentication activity 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.

audit_setup
: 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 next local log specified as the next path in the `/etc/sec/auditd_loc` (with `auditd –r`) or the default local path (`/var/adm`).

`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.

  For a description of the information contained in a report for `login` and other `auth_events`, see Section 10.11.2.2.

You may also define site-specific events for auditing. See Section 10.9 for more information on site-defined audit events.

10.11 Audit Reports

The trusted Digital UNIX operating system offers two programs to generate audit reports: `dxaudit` and `audit_tool`. `dxaudit` is a windows-based program that allows you to create selection files as well as the final report. The `audit_tool` program operates from the command line.

10.11.1 Generating Audit Reports with the dxaudit Program

The `dxaudit` program provides a graphical interface for the creation of audit reports.

The functions performed with the `XIsso` program have been moved to other GUIs. The `XIsso` program in this release is only an interface to the other GUIs and support for `XIsso` will be discontinued after this release.

10.11.1.1 Selection Files

The Modify Selection Files window found under the Reports menu item allows you to create and modify selection files. A selection file is used to select specific events for your audit reports. Only audit records that contain events and identification items that you specify are displayed. You can also specify a time range. Using selection files can speed up the review of your audit reports. See the online help for selection files for more information.

10.11.1.2 Deselection Files

The Modify Deselection Files window found under the Reports menu item allows you to create and modify deselection files. A deselection file is used to filter unwanted events from your audit reports. All audit records that contain events and identification items that you specify are suppressed. Any events listed in the deselection file take precedence over events in a selection file. Using deselection files can speed up the review of your audit reports. See the online help for deselection files for more information.

10.11.1.3 Reports

The Generating Reports window found under the Reports menu item allows you to generate audit reports. You can select the audit log to be reviewed as well as the selection and deselection files to be used as a filter. You can tailor the output format to your needs using this window. See the online help for reports for more information.

10.11.2 Generating Audit Reports with the audit_tool Program

The long format for an audit record returned by `audit_tool` is made up of information found in every audit record and of some information specific to the particular record.

The following information is common to every audit record:

- AUID, RUID, and EUID
- User name (if available)
- PID, parent PID
- Device where event occurred (major #, minor #)
- Event
- Result or error from event
- Host name on which event occurred
- Timestamp

The following is an example audit record.

```
1  audit_id:    1621      2 ruid/euid: 0/0  3 username: jdoe
4  pid:         5742      5 ppid: 1         6 cttydev: (39,0)
7  event:       login
   login name:  jdoe
   home dir:    /usr/users/jdoe
   shell:       /bin/csh
   devname:     tty02
8  char param:  Login succeeded
   char param:  ZK33C5
   directory:   /usr/users/jdoe
```

```
9  result:      0
10 ip address:  16.153.127.240 (alpha1)
11 timestamp:   Wed Jul 28 19:17:52.63 1993 EDT
```

The entries in every audit record are as follows:

1 The audit ID (AUID).

2 The real user ID (RUID) and the effective user ID (EUID).

3 The user's name. In the expanded (not the brief) report format, parentheses may be used around the user name. If the user name is not in parentheses, the user name is the name used at login time. If the user name is in parentheses, the name is the one associated with the RUID.

4 The process ID (PID).

5 The parent process ID (PPID). If a security violation occurs for an event that is the offspring of another event, the PPID allows you to trace back through a list of processes to the originating event, so you can learn the RUID and UID (and user) associated with the original event.

6 The device on which the event occurred. The record reports the major and minor numbers.

7 The event that was logged, such as `login` or `chmod`. This record reports information such as the login name and the home directory.

8 The parameter for the event; for example, if the event is the `chmod` system call, `char param` is the file name that was the argument for the system call.

9 If the event failed, the reason for failure. ''Permission denied'' is among the types of errors logged. For example, if a user attempted to access a file for which the user does not have permission, the audit report would contain the message ''error: Permission denied''.

10 The host name on which event occurred.

11 The timestamp for the event occurrence.

10.11.2.1 Audit Reports for System Calls

To extract records of `execve` and `open` system calls (the `-e` option) from the `/var/audit/auditlog.000` file in long format, enter the following command:

```
# /usr/sbin/audit_tool -e execve -e open -w /var/audit/auditlog.000
```

Example 10-5 shows a sample audit report for the `open` and `execve` system calls.

Example 10-5: Audit Report for System Calls

```
audit_id:    1621      ruid/euid: 1621/1621   (username: jdoe)
pid:         1304      ppid: 1249             cttydev: (6,2)
event:       execve
char param:  /sbin/cat
char param:  cat /etc/motd
vnode id:    8707   vnode dev:   (8,1024)    [regular file]
object mode: 0755
result:      0
ip address:  16.153.127.241 (alpha1)
timestamp:   Mon Aug 30 14:33:31.46 1993


audit_id:    1621      ruid/euid: 1621/1621   (username: jdoe)
pid:         1304      ppid: 1249             cttydev: (6,2)
procname:    /sbin/cat
event:       open
char param:  /etc/motd
flags:       0 : read
vnode id:    2284   vnode dev:   (8,1024)    [regular file]
object mode: 0644
result:      3 (0x3)
ip address:  16.153.127.241 (alpha1.sales.dec.com)
timestamp:   Mon Aug 30 14:33:31.46 1993

     .
     .
     .


8 records output
8 records processed
```

10.11.2.2 Audit Reports for Trusted Events

To extract records of the `auth_event` and `login` trusted events from the file `/var/audit/auditlog.000` and present them in long format, enter the following command:

```
# /usr/sbin/audit_tool –e auth_event –e login \
                                  /var/audit/auditlog.000
```

Example 10-6 shows a sample audit report for the `auth_event` and `login` trusted events.

Example 10-6: Audit Report for Trusted Events

```
audit_id: 1592         ruid/euid: 0/0         username: jdoe
pid: 597               ppid: 1                dev: (5,2)
event:       login
login name:  jdoe
home dir:    /users/jdoe
shell:       /bin/csh
devname:     /dev/tty04
char param:  Login succeeded
directory:   /users/jdoe
result:      0
ip_addr:     191.5.6.67 (alpha1.sales.dec.com)
timestamp:   Wed Jan  5 15:44:08.47 1994


audit_id: 1592         ruid/euid: 1592/0      username: jdoe
pid: 24247             ppid: 597              dev: (5,2)
event:       auth_event
login name:  jdoe
 ...........
remote/secondary identification data --
login name:  root
 ...........
char param:  su
directory:   /users/jdoe
result:      0
ip_addr:     191.5.6.67 (alpha1.sales.dec.com)
timestamp:   Wed Jan  5 15:44:25.24 1994
```

A remote system name is present if the login was done remotely. The remote system name is the LAT server name if the line is a LAT line. The remote system name is the host name if the login is by `rlogin`. There is always either an `error:` field or a `result:` field. It corresponds to the exit code that `login` exits with. The result is zero only when `login` was successful.

If `login` fails, the reason for the failure is given.

10.11.2.3 Audit Reports for Process IDs

To extract records of events with a process ID of 1304 from the file `/var/audit/auditlog.000` and present them in long format, enter the following command:

```
# /usr/sbin/audit_tool –p 1304 /var/audit/auditlog.000
```

Example 10-7 shows a sample audit report for the process ID of 1304.

Example 10-7: Audit Report for Process IDs

```
audit_id:    1621      ruid/euid: 1621/1621   (username: jdoe)
pid:         1304      ppid: 1249             cttydev: (6,2)
event:       execve
char param:  /sbin/cat
char param:  cat /etc/motd
vnode id:    8707   vnode dev:   (8,1024)    [regular file]
object mode: 0755
result:      0
ip address:  16.153.127.241 (alpha1.sales.dec.com)
timestamp:   Mon Aug 30 14:33:31.46 1993


audit_id:    1621      ruid/euid: 1621/1621   (username: jdoe)
pid:         1304      ppid: 1249             cttydev: (6,2)
procname:    /sbin/cat
event:       open
char param:  /etc/motd
flags:       0 : read
vnode id:    2284   vnode dev:   (8,1024)    [regular file]
object mode: 0644
result:      3 (0x3)
ip address:  16.153.127.241 (alpha1.sales.dec.com)
timestamp:   Mon Aug 30 14:33:31.46 1993
```

10.11.2.4 Abbreviated Audit Reports

To extract audit records about the `execve` and `login` commands from the `/var/audit/auditlog.000` file and present the report in abbreviated format (the `–B` option), enter the following command:

```
# /usr/sbin/audit_tool –B –e execve –e login /var/audit/auditlog.000
```

Example 10-8 shows a sample abbreviated audit report for the `execve` and `login` commands.

Example 10-8: Abbreviated Audit 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)
1234:1234:1234    2059   0x0         execve (/usr/bin/tset tset -I -Q)
1234:1234:1234    2060   0x0         execve (/usr/bin/hostname hostname)
1234:1234:0       2061   0x0         execve (/usr/bin/ps ps gax)
1234:1234:1234    2062   0x0         execve (/usr/bin/ls ls -l)
1234:1234:1234    2063   0x0         execve (/usr/bin/who who)
1234:1234:1234    2066   0x0         execve (/usr/bin/csh csh)
1234:1234:1234    2067   0x0         execve (/usr/bin/ls ls -a)
```

Example 10-8: (continued)

```
1234:1234:1234    2068   0x0         execve (/usr/bin/cat cat .cshrc)
1234:1234:1234    2069   0x0         execve (/usr/bin/from from)
1234:1234:0       2072   0x0         execve (/usr/bin/mail mail
                                                guest@alpha1)
1234:1234:0       2072   0x0         execve (/usr/lib/sendmail -sendmail
                                                  guest@alpha1)
1234:1234:0       2077   0x0         execve (/usr/sbin/mailq
                                                 /usr/sbin/mailq)
1234:1234:1234    2078   0x0         execve (/usr/bin/rwho rwho)
1234:1234:0       2079   0x0         execve (/usr/bin/ps ps gax)
1234:1234:1234    2082   0x0         execve (/usr/bin/dbx dbx -k /vmunix)
1234:1234:1234    2095   0x0         execve (/usr/sbin/netstat
                                            /usr/sbin/netstat)
1234:1234:1234    2096   (err 13)    execve (/usr/sbin/auditd)
1234:1234:1234    2097   (err 13)    execve (/usr/sbin/auditmask)
```

The column headings of the report indicate the following:

AUID:RUID:EUID
: The audit ID, real UID, and effective UID associated with the event.

PID
: The process ID number.

RES
: The result number. Refer to the reference page for the specific system call for information about the result number.

ERR
: The error code. For a list of error codes and their meanings, see the reference pages for `errno`(2).

EVENT
: The event and some arguments appear in the last column.

To extract records about the `execve` and `login` commands from the file `/var/audit/auditlog.000` and present the report in abbreviated format, but with the user name displayed (the `–w` option) instead of the ID number, enter the following command:

```
# /usr/sbin/audit_tool –wB –e execve –e login /var/audit/auditlog.000
```

Example 10-9 shows a sample abbreviated audit report for the `execve` and `login` commands as shown in Example 10-8, with user names displayed instead of ID numbers.

Example 10-9: Abbreviated Audit Report with User Names

```
USERNAME   PID   RES/(ERR)   EVENT
-----      ---   ---------   ----------------------------------------
root       2056   0x0        execve (/usr/sbin/rlogind rlogind)
root       2057   0x0        execve (/usr/bin/login login -p -h
                                      alpha1.sales.dec.com guest)
jdoe       2057   0x0        login  (guest)
jdoe       2057   0x0        execve (/bin/sh -sh)
jdoe       2058   0x0        execve (/usr/bin/stty stty dec)
jdoe       2059   0x0        execve (/usr/bin/tset tset -I -Q)
jdoe       2060   0x0        execve (/usr/bin/hostname hostname)
```

Example 10-9: (continued)

```
jdoe     * 2061   0x0        execve (/usr/bin/ps ps gax)
jdoe       2062   0x0        execve (/usr/bin/ls ls -l)
jdoe       2063   0x0        execve (/usr/bin/who who)
jdoe       2066   0x0        execve (/usr/bin/csh csh)
jdoe       2067   0x0        execve (/usr/bin/ls ls -a)
jdoe       2068   0x0        execve (/usr/bin/cat cat .cshrc)
jdoe       2069   0x0        execve (/usr/bin/from from)
jdoe     * 2072   0x0        execve (/usr/bin/mail mail guest@alpha1)
jdoe     * 2072   0x0        execve (/usr/lib/sendmail -sendmail
                                                  guest@alpha1)
jdoe     * 2077   0x0        execve (/usr/sbin/mailq /usr/sbin/mailq)
jdoe       2078   0x0        execve (/usr/bin/rwho rwho)
jdoe     * 2079   0x0        execve (/usr/bin/ps ps gax)
jdoe       2082   0x0        execve (/usr/bin/dbx dbx -k /vmunix)
jdoe       2095   0x0        execve (/usr/sbin/netstat
                                        /usr/sbin/netstat)
jdoe       2096   (err 13)   execve (/usr/sbin/auditd)
jdoe       2097   (err 13)   execve (/usr/sbin/auditmask)
```

The column headings are the same as in Example 10-8 except for USERNAME, which is the name associated with RUIDs using the `getpw*` routines. If the login user name is not available, the RUID is provided. Also, the asterisk (*) in front of the PID column indicates that RUID and EUID are different from each other.

10.12 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 `audit-data.`*n* in the crash directory, where *n* is the crash number. If no audit data was present, the `audit-data.`*n* 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 the `audit-data.`*n* file. It is also possible that the first audit record in `audit-data.`*n* may not be a complete record (`audit_tool` marks this as a corrupted record). In this case, the audit record has already been written to the `auditlog` file (or to the remote host).

10.13 Implementation Notes

The following is information about the Digital 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 vnode 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 10-3 for a list of the calls.
- Only TIOCSTI operations are audited for the `ioctl` system call.
- Only F_DUPFD and F_CNVT are audited for the `fcntl` system call.

10.14 Traditional UNIX Logging Tools

Although you are urged to use the Digital UNIX audit subsystem for security-relevant auditing, the traditional UNIX operating system logging tools 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 event 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 information about events on the system is stored in a number of different files.

Table 10-2 lists those files storing security-relevant information. The presence of specific log files on your system depends on which logging and

accounting features you have enabled.

Table 10-2: Traditional UNIX Log Files in /var/adm

File Name	Security-Relevant Information
`wtmp`	Records all logins, logouts, and shutdowns. You must 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.
`syslog.dated/`*date*`/auth.log`	Failed logins and system shutdowns.
`syslog.dated/`*date*`/syslog.log`	Requests for DECnet file transfers.

Because these files are your record of what has happened on the system, you should protect their contents. The files and directories should be owned by the root account and they should not be writeable by `group` or `other`.

For a complete discussion of the accounting software on your system, see the *System Administration* manual.

10.15 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 10.15.5 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.

10.15.1 Installing Audit

Audit must be configured into the kernel. You can either build a kernel with the DEC_AUDIT option, or run the `/usr/sbin/audit_setup` script.

The `audit_setup` script does the following:

- Establish startup flags for the audit daemon
- Establish startup flags for the auditmask
- Create `/dev/audit` (if needed)
- Configure a new kernel (if needed)

Startup flags are stored in `/etc/rc.config`. It is recommended that you select all the defaults, with the following exceptions:

- When asked to create `/var/audit`, select `y`.
- When asked to choose the action to take on an overflow, choose the option to overwrite the current auditlog.

Note

When prompted for event list, enter Return to have no events audited by default.

If a new kernel must be configured, audit is enabled when the new kernel is booted. If the kernel already supports audit, the `audit_setup` script will optionally enable audit.

See the `audit_setup`(8) reference page for further information.

10.15.2 Enabling Audit

There are two steps are required to enable audit:

1. Start the audit daemon (`auditd`)
2. Using the `auditmask` command, specify the events for audit. (For purposes of tracing, this may be best left to default to the null set of events at this point.)

Audit can be enabled using any one of the following approaches:

- `audit_setup` enables audit if the kernel supports it
- The `init` script enables audit according to the `RC.CONFIG` flags (which previously had been set by `audit_setup`) as follows:

```
# /sbin/init.d/audit start
```

- Manually starting `auditd`. For example, to collect data in the file `./FILE.`*nnn*(`-l`), return the location of data file (`-q`), and overwrite log on overflow condition (`-o o`) enter the following commands:

```
# auditd -ql FILE -l -o overwrite
```

To stop the collection of audit data, either clear the auditmask (for the system or any process whose auditmask you set), or turn off `auditd`:

```
# auditd -k
```

See the `auditd`(8) reference page for further information.

10.15.3 Tracing a Process

You can use the `auditmask` utility to adjust the audit characteristics of the system, of a single process, or all processes associated with a user's AUID.

Whether or not an event generates an audit record is a function of the system auditmask, the process auditmask, and the process `audcntl` flag. Each auditmask is a bitmap for all the events. The process `audcntl` flag specifies one of the following:

or	Audit if the event is in either the system or the process mask
and	Audit if the event is in both the system and the process mask
off	Do not audit this process
usr	Audit if the event is in the process mask

The default `audcntl` flag setting is `or`.

Specifying which events are audited is done by naming a set of system calls, alias names, or both. Aliases are defined in (and may be added to) `/etc/sec/event_aliases`. An optional extension can be used to distinguish between successful and failed occurrences of any event as follows:

```
open:1:0 specifies successful occurrences of open()
open:0:1 specifies failed occurrences of open()
```

The following examples demonstrate 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 args
```

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 exec'd `command` program:

```
# auditmask open:0:1 ipc:1:0 –e command args
```

In the following example, for PID 999, audit all (`–f`) events except `gettimeofday`:

```
# auditmask –p 999 –f gettimeofday:0:0
```

In the following example, for PID 999, audit no (`–n`) events:

```
# auditmask –p 999 –n
```

In the following example, for PID 999, audit `open` and `exec` events:

```
# auditmask –p 999 open exec
```

In the following example, for PID 999, add `exit` to the set of events being audited:

```
# auditmask -p 999 exit
```

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
```

In the following example, get abbreviated help for `auditmask`:

```
# auditmask –h
```

See the `auditmask`(8) reference page for further information.

10.15.4 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:

```
# auditmask –E date
Sun Nov 26 19:17:52 EST 1995

# audit_tool ‘auditd –dq‘ –B

AUID:RUID:EUID PID   RES/(ERR)     EVENT
-------------- ---   ---------     -----
1123:0:0       6691  0x14          audcntl (0x7 0x0 0x14 0x0)
1123:0:0       6691  0x0           execve (/sbin/date date)
1123:0:0       6691  0x2000        getpagesize ( )
1123:0:0       6691  0x2000        getpagesize ( )
1123:0:0       6691  0x0           getrlimit ( )
1123:0:0       6691  0x3ffc0004000 mmap (-1 0x7 0x3ffc0004000
                                     0x12 0x2000)
1123:0:0       6691  0x0           getrlimit ( )
1123:0:0       6691  0x3ffc0006000 mmap (-1 0x7 0x3ffc0006000
                                     0x12 0x4000)
1123:0:0       6691  0x0           getuid ( )
1123:0:0       6691  0x1           getgid ( )
1123:0:0       6691  0xa68         read ( 3 )
1123:0:0       6691  0x120000000   mmap (3 0x5 0x120000000
                                     0x102 0x4000)
1123:0:0       6691  0x140000000   mmap (3 0x7 0x140000000
                                     0x2 0x2000)
1123:0:0       6691  0x4           open (/shlib/libc.so 0x0)
1123:0:0       6691  0xa68         read (/shlib/libc.so)
1123:0:0       6691  0x3ff80080000 mmap (/shlib/libc.so 0x5
                                   0x3ff80080000 0x2 0x10e000)
1123:0:0       6691  0x3ffc0080000 mmap (/shlib/libc.so 0x7
                                   0x3ffc0080000 0x2 0x10000)
1123:0:0       6691  0x3ffc0090000 mmap (-1 0x7 0x3ffc0090000
                                     0x12 0x9bf0)
1123:0:0       6691  0x0           close ( 4 )
1123:0:0       6691  0x0           stat (/shlib/libc.so)
1123:0:0       6691  0x0           set_program_attributes ( )
1123:0:0       6691  0x0           close (3)
1123:0:0       6691  0x2000        getpagesize ( )
1123:0:0       6691  0x0           obreak (0x14000ea10)
1123:0:0       6691  0x0           gettimeofday ( )
1123:0:0       6691  0x3           open (
                                   /etc/zoneinfo/localtime 0x0)
1123:0:0       6691  0x32e           read (
                                   /etc/zoneinfo/localtime)
1123:0:0       6691  0x0             close ( 3 )
1123:0:0       6691  0x1d            write ( 1 )
1123:0:0       6691  0x0             close ( 0 )
1123:0:0       6691  0x0             close ( 1 )
```

```
1123:0:0       6691  0x0           close ( 2 )
1123:0:0       6691  0x0           exit ( )
```

```
# audit_tool ‘auditd –dq‘ –e exec

audit_id:    1123       ruid/euid: 0/0
pid:         6691       ppid: 6688            cttydev: (6,7)
event:       execve
char param:  /sbin/date
char param:  date
inode id:    7390       inode dev: (8,16384)  [regular file]
object mode: 0755
result:      0
ip address:  16.140.128.241 (alpha1.research.dec.com)
timestamp:   Sun Nov 26 19:17:52.77 1995 EST
```

Additional options are available to help select data of interest:

```
# audit_tool
Audit reduction tool usage: [options] logfile

Selection options:
   -a audit_id                -e event[.subevent][:succeed:fail]
   -E error# or error_string  -h hostname or ip address
   -p [-]pid                  -P ppid
   -r real_uid                -s string_parameter
   -t start_time              -T end_time - yymmdd[hh[mm[ss]]]
   -u uid                     -U username
   -v inode_id                -V inode’s device-major#,minor#
   -x device-major#,minor#    -y procname
   -/ search-string

Control options:
   -b        Output in binary format
   -B        Output in abbreviated format
   -d file   Use specified deselection rules file
              (-D to print ruleset)
   -f        Keep reading auditlog (like tail -f)
   -F        Fast mode; no state data maintained
   -i        Interactive selection mode
   -o        Override switching logfile due to change_auditlog
   -O format Output in specified format
         {cpu,usec,time,username,userid,pid,ppid,tid,res,event}
   -Q        Suppress progress messages
   -R [name] Generate reports by audit_id
   -S        Sort audit records by time (for SMP only)
   -w        Map ruid, group #s to names using passwd, group tbls
   -Z        Display statistics for selected events
```

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

10.15.5 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 ''c'' encoding indicates a file descriptor is recorded. The ''a'' encoding indicates an the integer argument is recorded. The set of encodings is described in the `<sys/audit.h>` file.

10.15.6 System Calls Not Always Audited

Not every instance of each system call generates audit data. The conditions under which a particular system call does not generate an audit record are described in Table 10-3.

Table 10-3: System Calls Not Always Audited

System Call	When Audit Record Is Not Generated
`close`	EBADF failures
`dup2`	Failures from `getf()`
`execv, execve, exec_with_loader`	Failures triggered by failed namei lookups, failure to terminate thread, abort from handler callout
`fcntl(*)`	Failures from `getf()`
`ioctl(*)`	Failures from `getf()`

Table 10-3: (continued)

System Call	When Audit Record Is Not Generated
`priocntlset()`	Failed input tests. Calls other than those which modify another process
`proplist_system()`	Failure on `copyin` of system call arguments
`reboot()`	Successful `reboot()`
`security()`	`getluid` option
`swapctl()`	Any call other than successful SC_ADD option
`uadmin()`	Any call other than a failed A_REBOOT or A_SHUTDOWN

All instances of any system call not in Table 10-3 generate audit data.

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.

When tracing a currently running process, use the `auditmask –c usr` option to trace all options for these system calls.

Administering ACLs 11

This chapter describes the installation and administration of the ACLs on a Digital UNIX system.

The Digital UNIX ACLs are based on the POSIX P1003.6 Draft 13 standard. The ACL library routines may change as the P1003.6 standard is finalized.

11.1 Digital UNIX ACLs Overview

The ACL subsystem is shipped as part of the base system, but is a kernel build-time option. Note that building the ACL subsystem is not the default choice. 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.

ACLs are currently supported on UFS, AdvFS, and NFS-mounted file systems. ACLs on DFS, CDFS and Internet IPC sockets are not currently supported.

Where the ACL is stored depends on the type of object it is associated with. ACLs for file system objects are stored in the property list associated with the object on the local Digital UNIX file system.

Because Digital UNIX ACLs are stored on a file system's property list, Digital UNIX supports only file systems that provide property lists. The currently supported files systems are as follows:

- Network file system (NFS)
- UNIX file system (UFS)
- Advanced file system (AdvFS)

See the `proplist`(4) reference page for more information about property lists.

See Section 21.8 for information on importing and exporting data between file systems.

11.2 Administration Tasks

The primary tasks of the administrator relative to the trusted system's access control list mechanisms are:

- 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.

To administer the ACLs on your Digital UNIX system, you need to be familiar with the commands documented on the following reference pages:

`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 ACL implementation

11.3 Installing ACLs

The optional Digital UNIX ACLs are shipped as part of the base system and can be loaded and used independently of other enhanced 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 `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 5.9 for a description of the ACL inheritance rules.

11.3.1 Enabling ACLs

The Digital UNIX ACL subsystem is enabled by editing the configuration file, rebuilding the kernel, and rebooting the system. Use the following procedure to enable ACLs.

1. Change directory to `/usr/sys`.
2. Use the following command to edit the machine's configuration file to add a line that has the string ''options DEC_ACL'':

   # **doconfig –c** *MACHINE*

3. Exiting the `doconfig` edit session automatically starts a kernel build.
4. Copy the new kernel to `/vmunix` using the following command:

   # **cp /usr/sys/***MACHINE***/vmunix /vmunix**

5. Reboot your machine in the normal fashion.

Example 11-1 illustrates how to enable ACLs.

Example 11-1: Enabling ACLs

```
# cd /usr/sys
# doconfig -c MACHINE

*** KERNEL CONFIGURATION AND BUILD PROCEDURE ***

Saving /usr/sys/conf/MACHINE as /usr/sys/conf/MACHINE.bck

Do you want to edit the configuration file? (y/n) [n]: y

Using ed to edit the configuration file.  Press return when ready,
or type 'quit' to skip the editing session:
2397
a
options DEC_ACL
.
w
2413
q


*** PERFORMING KERNEL BUILD ***
     Working....Tue Oct  3 08:29:47 EDT 1995
     Working....Tue Oct  3 08:31:50 EDT 1995

The new kernel is /usr/sys/MACHINE/vmunix
# cp /usr/sys/MACHINE/vmunix /vmunix
# reboot
```

11.3.2 Disabling ACLs

The Digital UNIX ACL subsystem is disabled by editing the configuration file, rebuilding the kernel, and rebooting the system. Use the following procedure to disable ACLs.

1. Change directory to `/usr/sys`.
2. Use the following command to edit the machine's configuration file to remove the ''options DEC_ACL'' string:

   # **doconfig –c** *MACHINE*

3. Exiting the `doconfig` edit session automatically starts a kernel build.
4. Copy the new kernel to `/vmunix` using the following command:

   # **cp /usr/sys/***MACHINE***/vmunix /vmunix**

5. Reboot your machine in the normal fashion.

Example 11-2 illustrates how to disable ACLs.

Example 11-2: Disabling ACLs

```
# doconfig -c MACHINE

*** KERNEL CONFIGURATION AND BUILD PROCEDURE ***

Saving /usr/sys/conf/MACHINE as /usr/sys/conf/MACHINE.bck

Do you want to edit the configuration file? (y/n) [n]: y

Using ed to edit the configuration file.  Press return when ready,
or type 'quit' to skip the editing session:
2395
/DEC_ACL
options DEC_ACL
d
w
q

Are you satisfied with the changes made during the editing
session? (y/n) [y]: y

*** PERFORMING Kernel Build ***
     Working....Tue Oct  3 08:41:10 EDT 1995
     Working....Tue Oct  3 08:43:13 EDT 1995

The new kernel is /usr/sys/MACHINE/vmunix
# cp /usr/sys/MACHINE/vmunix /vmunix
# reboot
```

11.3.3 Verifying Kernel Changes

To determine the status of ACLs in a kernel you are going to boot, enter the following command:

```
# nm –Bn /vmunix | grep sp_insert_ir
```

If the string ‘‘sp_insert_ir’’ is present, the ACLs are loaded in the kernel; if the string ‘‘sp_insert_ir’’ is not present, the ACLs are not loaded.

11.3.4 Determining If ACLs Are Enabled

Use the `dbx` command to determine if ACLs are currently running in the system as follows (make sure that `/vmunix` is the kernel that was booted):

```
# dbx /vmunix –k

dbx version 3.11.8
Type 'help' for help.


warning: cannot get register (number = 70)

stopped at  [thread_block:2016 ,0xfffffc00002b2da0] \
                        Source not available

warning: Files compiled -g3: parameter values probably wrong
(dbx) print paclpolicy
0
```

The output of the `print paclpolicy` is interpreted as follows:

0 ACLs are active.

1 ACLs are not active.

Note that `paclpolicy` is a global variable in the kernel.

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 present. This is accomplished by not shipping ACLs with the system and `fsck` and `fsdb` having ability to extract and manipulate ACLs from the property list obtained from the raw partition.

Ensuring Authentication Database Integrity 12

The information systems security officer (ISSO) is responsible for ensuring the integrity of the system. To do this, the ISSO runs the `authck` program, which checks the internal consistency of the files that make up the authentication database. (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 authentication database, consists of the following subsidiary databases:

- Protected password database (`/tcb/files/auth.db`, `/var/tcb/files/auth.db`, and `/tcb/files/auth/<`*a-z*`>/`*username)*
- 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.

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.

`-a` Performs the functions of `-f`, `-p`, and `-t`.

`-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.

Security Integration Architecture 13

This chapter describes the Security Integration Architecture (SIA) for Digital UNIX. The chapter discusses the following topics:

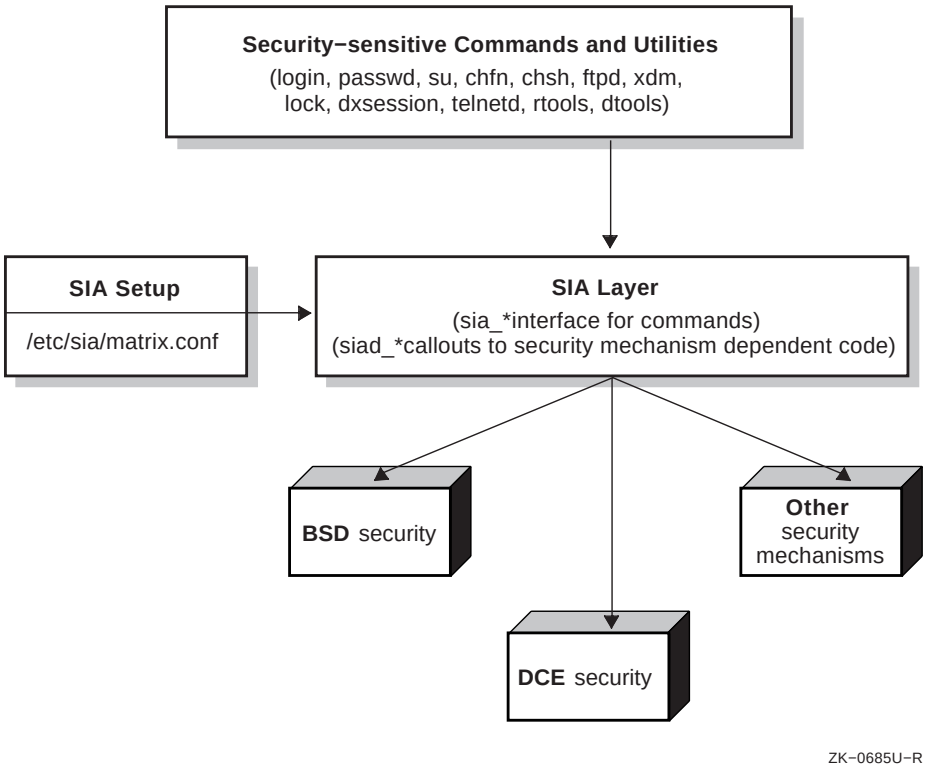
- Overview of the SIA
- Supported security configurations
- SIA's `matrix.conf` file
- Installation and deletion of layered security products

13.1 SIA Overview

All security mechanisms that run on the Digital 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 Digital UNIX with no modification to the security-sensitive Digital 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 Digital UNIX operating system currently 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 and audit capability.

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/bsd_matrix.conf`) and after the enhanced security subset is installed, an enhanced security `matrix.conf` file `/etc/sia/OSFC2_matrix.conf`). Each layered security product

provides its own `matrix.conf` file. The SIA layer looks for the `matrix.conf` file that is linked to the appropriate configuration file.

Note

Do not edit the `matrix.conf` file. The system administrator should only relink `matrix.conf` files.

Example 13-1 shows the default BSD `matrix.conf` (`/etc/sia/bsd_matrix.conf`) file:

Example 13-1: Default /etc/sia/bsd_matrix.conf File

```
#
# sia matrix configuration file (BSD only)
#
siad_init=(BSD,libc.so)
siad_chk_invoker=(BSD,libc.so)
siad_ses_init=(BSD,libc.so)
siad_ses_authent=(BSD,libc.so)
siad_ses_estab=(BSD,libc.so)
siad_ses_launch=(BSD,libc.so)
siad_ses_suauthent=(BSD,libc.so)
siad_ses_reauthent=(BSD,libc.so)
siad_chg_finger=(BSD,libc.so)
siad_chg_password=(BSD,libc.so)
siad_chg_shell=(BSD,libc.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=(BSD,libc.so)
siad_chk_user=(BSD,libc.so)
```

Example 13-2 shows the default enhanced security `matrix.conf` (`/etc/sia/OSFC2_matrix.conf`) file:

Example 13-2: Default /etc/sia/OSFC2_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)
```

Example 13-2: (continued)

```
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)
```

Example 13-3 shows the default DCE `matrix.conf` (`/etc/sia/dce_matrix.conf`) file:

Example 13-3: Default /etc/sia/dce_matrix.conf File

```
# sia matrix configuration file

siad_init=(DCE,/usr/shlib/libdcesiad.so),(BSD,libc.so)
siad_chk_invoker=(DCE,/usr/shlib/libdcesiad.so),(BSD,libc.so)
siad_ses_init=(DCE,/usr/shlib/libdcesiad.so),(BSD,libc.so)
siad_ses_authent=(DCE,/usr/shlib/libdcesiad.so),(BSD,libc.so)
siad_ses_estab=(DCE,/usr/shlib/libdcesiad.so),(BSD,libc.so)
siad_ses_launch=(DCE,/usr/shlib/libdcesiad.so),(BSD,libc.so)
siad_ses_suauthent=(DCE,/usr/shlib/libdcesiad.so),(BSD,libc.so)
siad_ses_reauthent=(DCE,/usr/shlib/libdcesiad.so),(BSD,libc.so)
siad_chg_finger=(DCE,/usr/shlib/libdcesiad.so),(BSD,libc.so)
siad_chg_password=(DCE,/usr/shlib/libdcesiad.so),(BSD,libc.so)
siad_chg_shell=(DCE,/usr/shlib/libdcesiad.so),(BSD,libc.so)
siad_getpwent=(DCE,/usr/shlib/libdcesiad.so),(BSD,libc.so)
siad_getpwuid=(DCE,/usr/shlib/libdcesiad.so),(BSD,libc.so)
siad_getpwnam=(DCE,/usr/shlib/libdcesiad.so),(BSD,libc.so)
siad_setpwent=(DCE,/usr/shlib/libdcesiad.so),(BSD,libc.so)
siad_endpwent=(DCE,/usr/shlib/libdcesiad.so),(BSD,libc.so)
siad_getgrent=(DCE,/usr/shlib/libdcesiad.so),(BSD,libc.so)
siad_getgrgid=(DCE,/usr/shlib/libdcesiad.so),(BSD,libc.so)
siad_getgrnam=(DCE,/usr/shlib/libdcesiad.so),(BSD,libc.so)
siad_setgrent=(DCE,/usr/shlib/libdcesiad.so),(BSD,libc.so)
siad_endgrent=(DCE,/usr/shlib/libdcesiad.so),(BSD,libc.so)
siad_ses_release=(DCE,/usr/shlib/libdcesiad.so),(BSD,libc.so)
siad_chk_user=(DCE,/usr/shlib/libdcesiad.so),(BSD,libc.so)
```

See the `matrix.conf`(4) reference page 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. Link the `/etc/sia/matrix.conf` file to the new `matrix.conf` file provided by the layered product using the `ln –sf new_matrix.conf matrix.conf` command.
4. Reboot your system.

13.5 Installing Multiple Layered Security Products

The Digital 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. Link the `/etc/sia/matrix.conf` file to the new `matrix.conf` file provided by the layered product using the `ln –sf new_matrix.conf matrix.conf` command. The product's installation procedure will provide details about the new `matrix.conf` files provided.
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 `rm /etc/sia/matrix.conf` command (the file that is linked is not removed).
4. Link the `/etc/sia/matrix.conf` file to the appropriate `matrix.conf` file. For example, `ln –s /etc/sia/bsd_matrix.conf /etc/sia/matrix.conf`.
5. Reboot your system.

Example 13-4 shows how to delete a layered security product and return to BASE security.

Example 13-4: Deleting a Layered Security Product

```
# /usr/sbin/shutdown now
# /sbin/rm /etc/sia/matrix.conf
# /sbin/ln –s /etc/sia/bsd_matrix.conf /etc/sia/matrix.conf
# /usr/sbin/reboot
```

Trusted System Troubleshooting 14

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.

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. For example, the system default database is stored in `/etc/auth/system/default`; its associated lock file is `/etc/auth/system/default:t`.

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`
- `/etc/auth/system/ttys`
- `/tcb/files/auth/<a-z>/`*username*

14.2 Invalid Maps

The system maintains two binary databases of mappings, one between user names and IDs and the other between group names and IDs. These are stored in the `/etc/auth/system` directory, in the files `pw_id_map` and `gr_id_map`.

Mapping names to IDs is a basic authentication and integrity mechanism of the system. If the underlying files on which the maps are based, `/etc/passwd` and `/etc/group`, are modified by trusted programs such as `passwd`, the programs automatically rebuild the binary databases. However, if the maps become inconsistent or their contents become corrupted, the system may not be able to map between names and IDs.

To prevent against corrupt mapping databases, the system removes the maps at startup. Because they are rebuilt every time the modification time of the underlying file is later than that of the database, the binary databases can be removed at any time. If the behavior of a program suggests that it cannot map a name to an ID, try removing the `/etc/auth/system/pw_id_map` or `/etc/auth/system/gr_id_map` file and running the failed program again. This procedure might avoid a system reboot.

14.3 Required Files and File Contents

The following files are required to run the system:

- `/tcb/files/auth.db` or `/tcb/files/auth/r/root`
- `/etc/auth/system/ttys.db`
- `/etc/auth/system/default`
- `/etc/auth/system/devassign`
- `/etc/passwd`
- `/etc/group`
- `/etc/auth/system/pw_id_map`
- `/etc/auth/system/gr_id_map`
- `/sbin/rc[023]s`
- `/dev/console`
- `/dev/tty*`
- `/dev/pty*`
- `/dev/ptm*`
- `/dev/pts/*`

- `/sbin/sh`
- `/vmunix`

14.3.1 The /tcb/files/auth/r/root File

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 to 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*.

If the protected password database 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 authentication profile for root by entering the following command:

```
# edauth root
```

The following example shows a typical authentication profile 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.

14.3.2 The /etc/auth/system/ttys.db File

The terminal control database must have a valid entry for the system console. There must not be a `:t` lock file associated with this database. 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:
```

14.3.3 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:
```

14.3.4 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:
```

14.3.5 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).

14.3.6 The /etc/group File

The `/etc/group` file is the group database. This file must be present and its format must be correct.

14.3.7 The /etc/auth/system/pw_id_map File

The `/etc/auth/system/pw_id_map` file is the user name to ID mapping database. This file must be consistent. The system rebuilds this file if it is not present.

14.3.8 The /etc/auth/system/gr_id_map File

The `/etc/auth/system/gr_id_map` file is the group name to ID mapping database. This file must be consistent. The system rebuilds this file if it is not present.

14.3.9 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.

14.3.10 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.

14.3.11 The /dev/pts/* and /dev/tty* Files

The `/dev/pts/*` and `/dev/tty*` files are pseudo terminal devices used for interprocess communication.

14.3.12 The /sbin/sulogin File

The `/sbin/sulogin` executable file allows restricting access in single user mode to those users with the root password.

14.3.13 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.

14.3.14 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.

14.4 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 OSFC2SEC400 subset, enter the following commands:

```
# cd /
# /usr/lbin/fverify < /usr/.smdb./OSFC2SEC400.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 a running centralized account management, see Section 9.3.

Part 3: Programmer’s Guide to Security

Introduction for Programmers 15

This chapter describes the implication of running trusted applications on a trusted Digital 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 new security system calls (section 2) and routines (section 3).

The `libsecurity.a`, `libsecurity.so`, `libaud.a`, and the `libaud.so` libraries hold all new enhanced security interface binaries. Use the `-l` 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 Digital UNIX security interfaces. Following traditional UNIX practice, all Digital 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 Digital 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 Digital 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

Digital UNIX defines several directories to hold its security information. You can review the reference pages for a description of these files and directories, primarily section 4.

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 System Calls and Library Routines with Enhanced Security

The tables in the following sections list many of the Digital 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†, write`
Process control	`fork, sigpause, execve, sigsetmask, setpgrp†, sigvec, sigblock`
File attributes	`access, chroot†, chmod†, stat, chown†, umask`
User and group ID	`getegid, getuid, getgid, setgroups†, geteuid, setreuid†`
Auditing	`audcntl†, audgen†`
General	`syscall`

Table note:

† 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 Digital UNIX library routines that have security implications.

Table 15-3: Security-Relevant Library Routines

Category	Library Routines
File control	`fopen, popen`
Password handling	`getpass, putpwent, getpwnam, setpwent, getpwent, endpwent, getpwuid, passlen, pw_mapping, randomword, time_lock`
Process control	`signal`
Group processing	`Getgrent, setgrent, getgrnam, endgrent, getgrgid`
Identifying the user	`cuserid, getpwuid, getlogin`
Password encryption	`crypt, encrypt`
User and group ID	`setuid, setgid, setegid, setrgid, seteuid, setruid`
Authorization	`getesdvent, getesdfent, getesfient, getespwent, getestcent`

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 pseudo users and pseudo groups for this purpose.

- 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

  Digital 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.

Trusted Programming Techniques 16

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.

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. This information can help the program decide how to respond, or produce a more helpful diagnostic message. 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 Digital 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 a DECwindows Environment

The following sections discuss several ways to increase security in a DECwindows programming environment:

- 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.

DECterm windows provide 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 a DECterm 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.

DECterm windows block 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:

- A client selects the event for a window by setting its event mask
- A 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. 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.

Authentication Database 17

The authentication database is a set of databases that store all Digital UNIX security information. 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 you create often need to use the information in these databases. Except for a few specialized cases, system administrators maintain these databases using the Digital 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 – other documentation describes their administrative management.

17.1 Accessing the Databases

Digital 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 in conjunction with this chapter.

Subject	Reference Page
Device Assignment	`getesdvent`(3)
File Control	`getesfient`(3)
System Default	`getesdfent`(3)
Protected Password	`getespwent`(3)
Terminal Control	`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

A database consists of an ordered set of named entries. Programs primarily use the name of the entry to request a specific entry, although a program can also sequentially search through the entries.

Each entry contains an ordered set of fields. Each field has a name, used to access the field and a value. Each database has a set of fields that are allowed to be present in one of its entries. Individual fields are optional and can be omitted from an entry.

In general, library routines read or write the entry as a whole. Each such library defines a C structure to hold all possible fields for a given entry of the database. This structure is always accompanied by a mask that designates which fields are to be read or written.

Fields can hold many kinds of values (integer, string, and so forth). However, you only have to deal with the types in the C structure defined for reading and writing entries in each database. The actual format of the files of the database is neither visible nor needed.

Digital UNIX programs understand what to do when a field is undefined. Similarly, your programs should take some appropriate action. A particular case is when some undefined fields are fetched from the system defaults database, as described in the following section. 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 referenced from the same structure that stores values for the individual entry.

17.2.1 Database Form

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 entry 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:
```

Description	Identifier	Entry	Value
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 seconds	

Attempts since last login	t_failures#3	3 trys	
Account status	t_lock	@	Unlocked
Check entry		chkent	End of entry

The following C structure is used for fetching an entry (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 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 protected by a protected subsystem pseudogroup, 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 creation of database files is controlled by entries in the file control database. All database file attributes are stored there. The procedure for creating database entries is to create a new file that stores the database contents and then rename that file to replace the old one once the database has been rewritten. 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.

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. A common programming mistake is to read a database entry, change one or more field and flag values, and submit the same buffer to the routine that modifies the database.

Database entries come in two types; those that are self-contained and those that contain pointers to variable-length fields. The `getesdvent`(3) reference page describes the `copyesdvent()` routine that allocates a structure to hold a device assignment database entry and copy 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 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 a `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 all system defaults. 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 have defaults.

When your program reads an 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. System defaults are defined in normal operation, so if a system default field is also undefined, your program should refuse the request and issue an audit report to report the error, although in some cases you can use some default value when neither is defined.

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 a `es_term` entry and then call `putestcnam()` to set system defaults.

17.3 Device Assignment Database

The device assignment database contains device attributes for devices on the system. There are two kinds of devices:

- 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 entries in 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 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

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 install all new security-sensitive files and directories in the database. Include all of the attributes that do not change. This ensures that these attributes are regularly checked and corrected.

The file control database is a text file: `/etc/auth/system/files`. Use any text editor to add to or alter existing entries. See the `files`(4) reference page for a definition of the format of this file. 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 Digital UNIX convention is to append a `:t` to a pathname for the file's new contents) and then rename (using the `rename()` system call) the new file to the existing file.

You 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`, 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 protected password, device assignment, command authorization, 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.

17.6 Protected Password Database

The protected password database (`/tcb/files/auth.db` and `/var/tcb/files/auth.db`) holds a set of user authentication profiles. Each authentication profile 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. 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 `/etc/passwd` file is `getpwent()`. The interfaces' functions are unchanged and always fetch their information from the `/etc/passwd` file. 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 this database. However, many trusted programs need to read the information from the authentication profile.

Each authentication profile is held in a file whose name is the profile (user) name. The `/tcb/files/auth` directory contains a subdirectory for each each letter of the alphabet, and each of those subdirectories holds the files for accounts whose name begins with that letter. Each entry is rewritten individually. Only a single entry need be locked for the duration of a database update.

17.7 Terminal Control Database

The terminal control database, `/etc/auth/system/ttys.db`, 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.

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 this database, although the Digital 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.

Identification and Authentication 18

This chapter discusses the following topics:

- New authentication routines
- The audit ID and some guidelines for using it
- The support libraries
- Using daemons
- The user authentication profile in the protected password database
- Some brief cautions for handling passwords

18.1 New libsecurity Library Routines

In order to allow adding new fields to the current for getting and setting authentication information without violating binary compatibility, a new series of interfaces has been added to enhanced security. Take for example the user profile:

```
struct pr_passwd {
    struct pr_field ufld;
    struct pr_flag uflg;
    struct pr_field sfld;
    struct pr_flag sflg;
};
```

Any new field in `pr_field` changes the offsets to `uflg`, `sfld`, and `sflg` (even if you add to the end). Because the offsets are known and compiled into calling applications, this breaks binary compatibility.

Additions to (at least) `pr_passwd` are expected as new features are added.

18.1.1 Changed Application Programming Interfaces

Several of the security routines are being replaced in the Digital UNIX Version 4.0 release. Table 18-1 lists the affected interfaces and the replacements.

Table 18-1: Changed Programming Interfaces

Obsolete Interface	Replacement Interface
getprpwnam()	getespwnam()
putprpwnam()	putespwnam()
getprpwent()	getespwent()
getprpwuid()	getespwuid()
getprtcent()	getestcent()
getprtcnam()	getestcnam()
putprtcnam()	putestcnam()
getdvagnam()	getesdvnam()
getdvagent()	getesdvent()
copydvagent()	copyesdvent()
putdvagnam()	putesdvnam()
getprdfent()	getesdfent()
getprdfnam()	getesdfnam()
putprdfnam()	putesdfnam()
getprfient()	getesfient()
getprfinam()	getesfinam()
putprfinam()	putesfinam()
read_pw_fields()	[escap_parse_fields(espw_parse, ...)]
store_pw_fields()	[escap_print_fields(espw_parse, ...)]
read_tc_fields()	[escap_parse_fields(estc_parse, ...)]
store_tc_fields()	[escap_print_fields(estc_parse, ...)]
time_lock()	time_lock_es()
get_seed()	get_seed_es()
auth_for_terminal()	auth_for_terminal_es()
locked_out()	locked_out_es()

The following new APIs are added in the Digital UNIX Version 4.0 release:

```
copyespwent()
copyestcent()
copyesfient()
copyesdfent()
escap_parse_fields()
escap_print_fields()
escap_cmp_fields()
escap_copy_fields()
escap_size_data()
get_num_crypts()
get_crypt_name()
```

Table 18-2 lists the data structures changed in the Digital UNIX Version 4.0 release.

Table 18-2: Changed Data Structures

Obsolete Structures	Replacement Structures
pr_field	espw_field
pr_flag	espw_flag
pr_passwd	es_passwd
t_field	estc_field
t_flag	estc_flag
pr_term	es_term
dev_field	esdev_field
dev_flag	esdev_flag
dev_asg	esdev_asg
f_field	esfi_field
f_flag	esfi_flag
pr_file	es_file
system_default_fields	es_default_fields
system_default_flags	es_default_flags
pr_default	es_default

18.1.2 What to Do With Existing Programs

The obsolete routines with the old names and calling sequences are currently supported. These obsolete routines were rewritten to be layered on top of the new interfaces, so that the routines (`putprpwnam` for example) do not lose new fields, but will instead refetch the affected record, and propagate changes to the new-format interface, thereby preserving new fields.

Statically-compiled programs using the obsolete interfaces will lose or destroy the new fields if they are allowed to write data.

18.1.3 What to Do For New Programs

The new `libsecurity` library with the new extended routines needs to be compiled ahead of the application. New programs should use the new `getes*()` and `putes*()` routines.

18.2 The Audit ID

Digital UNIX preserves all traditional UNIX process user and group identities. Additionally, it provides the per-process audit ID (AUID), which is unique to Digital 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.

The `getluid()` and `setluid()` system calls read and set the audit ID. See their reference pages for details.

18.3 Identity Support Libraries

The Digital UNIX operating system provides several library routines for managing user and group identities. For example, the `set_auth_parameters()` routine, usually called at the beginning of a program's `main()` routine, stores the initial user and group IDs that can later be queried or tested by the other routines.

Several of the 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.

18.4 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, and then either perform the actions directly or execute a program to perform them.

18.5 Using the Protected Password Database

Although the protected password database is intended mainly for Digital 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 correspond to the user name and ID in `/etc/passwd`.

- Encrypted password (`u_pwd`)

  This is the real encrypted password. The system supports a longer password length than the traditional UNIX limit of 8 characters. The system encrypts the password in 8 character segments. Each cycle of encryption provides an 11-character value plus the initial 2-character salt string. Thus, the length of the encrypted password is the number of segments times 11 plus 2 ((11 * n) + 2).

  For example, a plain text password of 1 to 8 characters encodes to 13 characters. A plain text password of 9 to 16 characters encodes to 24 characters and a plain text password of 17 to 24 characters encodes to 35 characters.

  See the `crypt`(3) reference page for more information.

- Retired status (`u_retired`)

  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, in conjunction 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 password length in characters (`u_maxlen`)
  – 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`)

- 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 login.

- 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 locked due to too many unsuccessful attempts.

- Number of unsuccessful login attempts allowed before lock (`u_maxtries`)

  This value is the user-specific limit for the number of unsuccessful attempts allowed until the account locks.

- 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 only shows the status of the administrative lock. An account may also be locked due to a password lifetime expiration or exceeding the number of unsuccessful attempts allowed for the account.

Your program can assume that the user name and ID in the protected password database is maintained by the system to have a corresponding entry in the `/etc/passwd` file.

18.6 Example: Password Expiration Program

The program named `myexpire` in Example 18-1 prints the user's password expiration time as defined in the 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;


   /*--- 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 protected password database files are accessible only to processes in the `auth` group. Programs that need to read the protected 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 a user ID and have a group ID of `auth`.

18.7 Password Handling

Digital UNIX has been designed so that trusted programs can authenticate their users without specifically asking for passwords. Digital UNIX explicitly uses the audit ID for this purpose. Additional password handling is usually not necessary and difficult to handle securely. Appendix D provides an example of a program for password checking.

Audit Record Generation 19

This chapter presents an overview of the audit concerns from a programmer's perspective. The following subjects are discussed:

- Auditable events
- Disabling auditing for the current process
- Modifying what gets audited for the current process
- Generation of application-specific audit records
- Tokens

19.1 Categories of Auditable Events

Auditable events are divided into the following categories:

- System calls
- Trusted events (user events such as `login`)
- Site-defined events

A list of the default auditable events can be found in the Appendix B or by looking at the `/etc/sec/audit_events` file as delivered on your system. You can use the `audit_setup` script to establish the auditable events on your system. See the `audit_setup`(8) reference page for more information.

19.2 Generation of Audit Records

Whether an auditable event actually results in the generation of an audit record depends on the following:

- Process audit control flag
- System audit mask
- Process 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 gets generated if the process audit mask indicates such an event should be audited.

The process audit control flag also has two non-exclusive states:

AUDIT_SYSCALL_OFF	Turns off system call record generation for the application. The system calls specified in the system mask continue to be audited.
AUDIT_HABITAT_USR	Allows turning on the habitat system calls in the user mask for an application while system calls are turned off for the system mask. See Appendix B for the habitat events.

A default audit level can be established for most users, while being able to single out specific users to audit more or less closely. A privileged process is allowed to specify what gets audited for itself either absolutely, or relative to the system audit mask.

Control over what events get audited is an important step in fine-tuning the audit data to indicate only events of interest. Sometimes too much audit data is generated by system calls. For example, the login process executes thousands of system calls, but a single audit record for the login process is easier to understand, less stressful on the system, and fewer entries in the auditlog. Modification of a security-relevant database, such as the password file, requires the auditing of information not easily picked up from the system calls, such as which field of which entry is being modified.

19.3 Disabling Auditing

Use the `audcntl()` system call to disable auditing for the current process as follows:

```
audcntl ( SET_PROC_ACNTL, (char *)0, 0, AUDIT_OFF, 0, 0 );
```

19.4 Modifying Process Audit Attributes

Control over what is audited for a specific process is achieved by modifying the process' `auditmask` and `audcntl` flags. Modify the process' audit

mask as follows:

```
/* ex. set the process' 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 (found in <sys/audit.h>), takes the following arguments:

buf The buffer in which the mask gets built

event name The <sys/audit.h> header file contains application event names. The <sys/*syscall.h> header files contains system call names.

succeed Indicates whether to audit successful occurrences of the event. A 1 means audit success.

fail Indicates whether to audit failed occurrences of the event. A 1 means audit failure.

See the audcntl(2) reference page for more information.

19.5 Audit Records and Tokens

The audit subsystem is designed to make it easy to add application-specific auditing.

The audit subsystem has no fixed record type. Instead, the various possible elements of an audit record are tuples consisting of defined audit tokens and corresponding values. Each audit record is then built up from those elements. This approach provides flexibility in the kind and amount of data that can go into an audit record. It also avoids the problems associated with a less flexible design that defines a number of fixed audit record types and requires all software that uses the audit subsystem to work within those record types.

Most tokens specify an audit value that is fixed in length and allow four bytes for the value. Examples of fixed-length tokens are AUD_T_AUID (audit ID), AUD_TP_DEV (device number), and AUD_TP_PID (process ID number).

Some types of audit values are of variable length. These are accommodated by tokens that are pointer types. AUD_T_CHARP (character parameter) and AUD_T_HOMEDIR (home directory) are two of the pointer-type tokens.

Some pointer-type tokens use iovec structures. Examples of these are AUD_T_OPAQUE, and AUD_T_INTARRAY which is used for data in integer arrays. Opaque data is displayed as an alphanumeric or in octal format. The iovec structure is defined in <sys/uio.h>. For information about iovec see the reference pages for readv(2) and writev(2).

19.5.1 Public Tokens

The tokens available to programs using the audgen() and audgenl() routines are called **public tokens**, and are named AUD_T_*name* in the <sys/audit.h> file. Example 19-1 lists the public tokens:

Example 19-1: Public Tokens

```
   /* start of ptr token types */
AUD_T_CHARP        character string
AUD_T_SOCK         socket
AUD_T_LOGIN        login name
AUD_T_HOMEDIR      home directory
AUD_T_SHELL        shell
AUD_T_DEVNAME      device name
AUD_T_SERVICE      service string (rfu)
AUD_T_HOSTNAME     host name
AUD_T_INTP         integer array (deprecated)
   /* 1st element of int array is # of data elements in array */

   /* start of iovec style data */
AUD_TOKEN_IOVEC_MIN
AUD_T_OPAQUE       opaque data
AUD_T_INTARRAY     integer array
AUD_T_GIDSET       group set
AUD_T_XDATA        aud_xdata structure
AUD_TOKEN_PTR_MAX
   /* end of iovec style data */
   /* end of ptr token types */

AUD_T_AUID         audit id
AUD_T_RUID         real uid
AUD_T_UID          effective uid
AUD_T_PID          pid
AUD_T_PPID         ppid
AUD_T_GID          gid
AUD_T_EVENT        event #
AUD_T_SUBEVENT     subevent #

AUD_T_DEV          dev major,minor #
AUD_T_ERRNO        errno value
AUD_T_RESULT       result value
```

Example 19-1: (continued)

```
AUD_T_MODE          object mode
AUD_T_HOSTADDR      ip address
AUD_T_INT           integer data
AUD_T_DESCRIP       descriptor
AUD_T_HOSTID        hostid

AUD_T_X_ATOM        X atom
AUD_T_X_CLIENT      X client identifier
AUD_T_X_PROPERTY    X property identifier
AUD_T_X_RES_CLASS   X resource class
AUD_T_X_RES_TYPE    X resource type
AUD_T_X_RES_ID      X resource identifier
```

Only the public tokens can be added to records from user space using the `audgen` call; the private tokens cannot.

Further, the public tokens are broken down into 3 categories:

pointer types	Represent data strings or structures
iovec style data	Added as iovec-formated data
default	32 or 64-bit quantities (AUD_T_RESULT is 64-bit; others are 32-bit)

19.5.2 Private Tokens

Tokens with security implications that are not available to users are called **private tokens**, and are named `AUD_TP_`*name*.

Example 19-2 lists the private tokens:

Example 19-2: Private Tokens

```
AUD_TP_ACCRGHT    msghdr's access rights data (sendmsg, recvmsg)
AUD_TP_MSGHDR     msghdr's name field (sendmsg, recvmsg)
AUD_TP_EVENTP     event name (for alternate habitats)
AUD_TP_HABITAT    habitat name
AUD_TP_ADDRVEC    address vectors (exportfs)
AUD_TP_INTP       integer array

AUD_TP_AUID       audit id
AUD_TP_RUID       real uid
AUD_TP_UID        effective uid
AUD_TP_PID        pid
AUD_TP_PPID       ppid
AUD_TP_HOSTADDR   host address (ip)
AUD_TP_EVENT      event #

AUD_TP_SUBEVENT   subevent #
AUD_TP_NCPU       cpu #
AUD_TP_DEV        device #
AUD_TP_LENGTH     audit record length
```

Example 19-2: (continued)

```
AUD_TP_IPC_GID    sysV structure gid value
AUD_TP_IPC_MODE   sysV structure mode bits
AUD_TP_IPC_UID    sysV structure uid value
AUD_TP_TV_SEC     timestamp (sec)

AUD_TP_TV_USEC    timestamp (usec)
AUD_TP_SHORT      short data
AUD_TP_LONG       long data
AUD_TP_VNODE_DEV  device on which inode resides
AUD_TP_VNODE_ID   inode identifier
AUD_TP_VNODE_MODE object mode
AUD_TP_VERSION    version #
AUD_TP_SET_UIDS   flag indicating a uid change occurred

AUD_TP_CONT       continuation flag
AUD_TP_TID        thread identifier (address)
```

19.6 Application-Specific Audit Records

The following are examples of code to generate an application-specific audit record:

```
/* ex. generate an event of type ‘event‘, with "some string",
   and a result of 66
*/
#include <sys/audit.h>

if ( audgenl ( ‘event‘, AUD_T_CHARP, "some string", \
                        AUD_T_RESULT, 66L, 0 ) == -1 )
   perror ( "audgenl" );
```

The event can be either a trusted event (see `audit.h`) or an administratively defined site-specific event.

```
/* ex. generate audit record for administratively defined event
   event is "rdb", subevent is "commit"
   note: this event must be registered in /etc/sec/site_events
   link with -laud
*/

#include <sys/audit.h>
int eventnum, subeventnum;

if ( aud_sitevent_num ( "rdb", "commit", &eventnum, \
                                    &subeventnum ) == -1 )
   return(-1);

if ( audgenl ( eventnum, AUD_T_SUBEVENT, subeventnum, \
                           AUD_T_CHARP, "whatever", 0 ) == -1 )
        return(-1);
```

See the `audgenl`(3) and `aud_sitevent`(3) reference pages for more information.

The examples in this section generate audit records only if the respective event types are selected to be audited and the auditable event actually results in the generation of an audit record (see Section 19.2).

Using the SIA Interface 20

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 Digital UNIX operating system. The SIA configuration framework isolates security sensitive commands from the specific security mechanisms. These 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.

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`	Spwans 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

Figure 20-1: SIA Layering

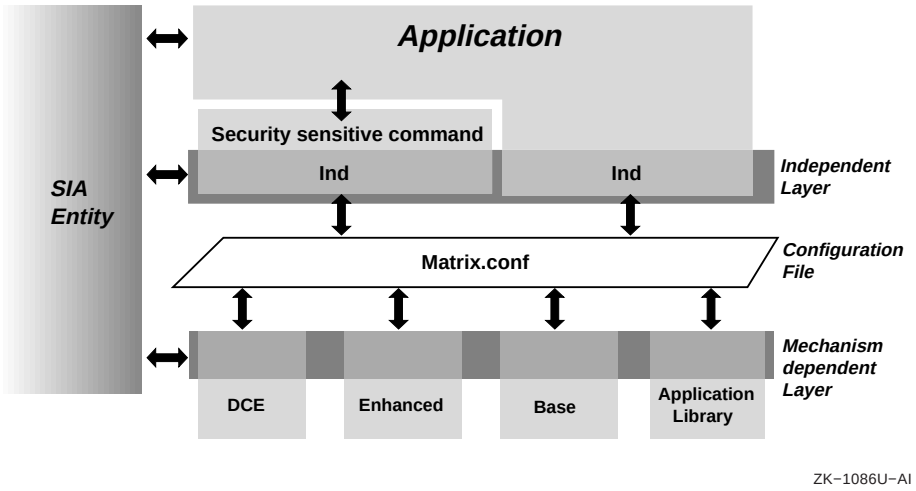


Table 20-2 and Table 20-3 list the SIA porting routines.

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
`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()`	Verifys 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 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 Digital 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 Digital UNIX that is either creating a new security mechanism or planning to use a specific security mechanism, requires SIA integration.

The SIA components consist of only user-level modules. The components resolve the configuration issues with respect to command and utility 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 Digital UNIX consists of the following two groups of interface routines:

`sia_*()` The commands and utilities porting interface

`siad_*()` The security mechanism-dependent porting interface

Security mechanisms deliver a shared library containing the `siad_*()` routines and provide 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 security-sensitive commands have been modified to use the mechanism-independent `sia_*()` routines. The routines are used by the commands and utilities to access security functions yet remain isolated from the specific security technologies. The `sia_*()` routines call the associated mechanism-dependent `siad_*()` routine, 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-dependent layer. The calling process uses shared library functions to access and lookup 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. Digital UNIX uses the `dlopen()` and `dlsym()` shared library interfaces to open the specified security-mechanism shared library and lookup 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 `linc`. See Appendix E 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 a 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 is 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  */
                           /* 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 previous 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 command or utility is responsible for

providing the appropriate display characteristics.

The parameter collection capability provided by SIA uses the following interface definition in `sia.h`:

```
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 */

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 Digital

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 a `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 Audit Logs

SIA supports a general 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 20-2 is a typical `/var/adm/sialog` file.

Example 20-2: Typical /var/adm/sialog File

```
SIA:EVENT Wed Feb  3 05:21:31 1995
Successful SIA initialization
SIA:EVENT Wed Feb  3 05:22:08 1995
Successful session authentication for terry on :0
SIA:EVENT Wed Feb  3 05:22:08 1995
Successful establishment of session
SIA:ERROR Wed Feb  3 05:22:47 1995
Failure to authenticate session for root on :0
SIA:ERROR Wed Feb  3 05:22:52 1995
Failure to authenticate session for root on :0
SIA:EVENT Wed Feb  3 05:22:59 1995
Successful session authentication for root on :0
SIA:EVENT Wed Feb  3 05:22:59 1995
Successful establishment of session
SIA:EVENT Wed Feb  3 05:23:00 1995
Successful launching of session
SIA:EVENT Wed Feb  3 05:24:40 1995
Successful authentication for su from root to terry
SIA:EVENT Wed Feb  3 05:25:46 1995
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 be ordered after those that do not.

Notes

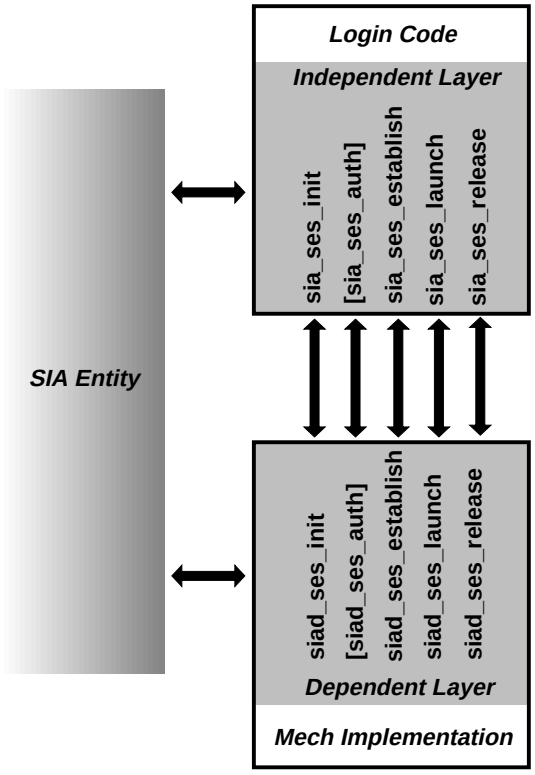
The default security mechanism, BASE, accepts authentication vouching.

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 of 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 for 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 Session Processing



ZK−1085U−AI

The session processing interfaces to the security mechanism-dependent routines (`siad_*()`) all 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 or 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 is the first and last stage 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 20-3 is a code fragment that shows session processing for the `login` command.

Example 20-3: Session Processing Code

```
          .
          .
          .
/* SIA LOGIN PROCESS BEGINS */

/* Logging of failures to sia_log is done within the libsia */
/* Logging to syslog is responsibility of calling routine   */

if((sia_ses_init(&entity, oargc, oargv, hostname, loginname, \
               ttyn, 1, NULL)) == SIASUCCESS) {

/***** SIA SESSION AUTHENTICATION *****/

   if(!fflag) {
      for(cnt=5; cnt; cnt--) {
```

Example 20-3: (continued)

```
         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) &&
       (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;
```

Example 20-3: (continued)

```
            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 *****/
    }
/***** 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 can not 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 re-collecting 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 on 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 the login process becoming 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 20-3 is from the `login` command.

20.12.6.2 The rshd Process

Session processing for `/usr/sbin/rshd` differs from `login`. The `rshd` process requires calling `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 `–f` 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 only provide one routine for both reentrant and non-reentrant, 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 works 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 Digital 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 this chapter. 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 Digital 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 relinking of a SIA `matrix.conf` file is followed by a reboot. System administrators must never be required to edit a live `matrix.conf` file.

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 simply 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 mechanism in Digital 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 staticly linked binaries found in `/sbin`. This can be accomplished by providing a `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, while the `siad_*()` convention is the strong symbol entry point that is actually used. See Appendix E for more information about routine naming conventions.

Programming With ACLs 21

This chapter discusses the following topics:

- Terms used for DAC read, write, execute, search, and owner access permissions
- ACL data representations
- Specifying default ACLs
- Rules for creating and replicating ACLs
- An example of ACL inheritance

Note

The ACL library routines (`-lpacl`) are not thread safe.

The Digital UNIX ACLs are based on the POSIX P1003.6 Draft 13 standard. The ACL library routines may change in the final version of the standard.

21.1 Introduction to ACLs

Aaccess control lists (ACLs) are the method used to implement discretionary access control (DAC). ACLs provide a more granular discretionary access control mechanism than traditional UNIX permission bits. All objects are considered to be protected by an ACL. An object protected by traditional UNIX DAC has a base entry ACL. If additional entries are added to the ACL, they are evaluated according to entry type by rules defined by the POSIX specification.

The POSIX ACL policy has been defined by the POSIX P1003.6 DAC security specification and features the following:

- An unordered ACL with a dependent relationship on the UNIX discretionary access control mechanism
- Inheritance rules that allow the presence of a default ACL on directories

In addition to the POSIX ACLs, Digital UNIX provides enhanced library interfaces.

You can apply ACLs to any traditional UNIX file system object that has permission bits that are used for access control, such as files and directories. Symbolic links have permission bits that are initialized according to the usual rules but are never used for access control; ACLs on this file type are not supported. The guidelines in this chapter apply to all objects that may be protected by ACLs.

Refer to the `acl`(4) reference page for a description of POSIX ACL support.

21.2 Library Routines

Table 21-1 lists the ACL library routines:

Table 21-1: ACL Library Routines

Routine	Description
`acl_add_perm( )`	Add a set of permissions
`acl_clear_perm( )`	Clear the permissions
`acl_convert_posix( )`	Remove non-POSIX entries from an ACL
`acl_copy_entry( )`	Copy an ACL entry
`acl_copy_ext( )`	Copies an ACL from system to user space
`acl_copy_int( )`	Copies an ACL from user to system space
`acl_create_entry( )`	Create a new ACL entry
`acl_delete_def_fd( )`	Delete the default ACL from a directory
`acl_delete_def_file( )`	Delete the default ACL from a file
`acl_delete_entry( )`	Delete an entry
`acl_delete_perm( )`	Delete a set of permissions
`acl_dup( )`	Create a duplicate ACL
`acl_first_entry( )`	Move current entry to first entry
`acl_free( )`	Free the ACL working storage memory
`acl_free_qualifier( )`	Free the acl_id entry in an ACL
`acl_free_text( )`	Free the ACL text working storage memory
`acl_from_text( )`	Convert the text ACL to the internal representation
`acl_get_entry( )`	Get the descriptor to an ACL
`acl_get_fd( )`	Get ACL using the file descriptor
`acl_get_file( )`	Retrieve an ACL
`acl_get_permset( )`	Retrieve the permissions for an ACL
`acl_get_qualifier( )`	Retrieve the ID from the current ACL
`acl_get_tag_type( )`	Retrieve the entry type identifier
`acl_init( )`	Allocate and initialize ACL working storage
`acl_locate_entry( )`	Locate an entry in an ACL
`acl_set_fd( )`	Set the ACL by the file descriptor
`acl_set_file( )`	Set the ACL by the pathname
`acl_set_permset( )`	Set permissions in a ACL entry
`acl_set_qualifier( )`	Set the ACL entry type

Table 21-1: (continued)

Routine	Description
`acl_set_tag_type( )`	Set the ACL entry type
`acl_size( )`	Determine the size of an ACL
`acl_to_text( )`	Convert an IR to an ASCII string
`acl_validate_and_sort( )`	Check ACL for validity and sort for decision order

21.3 Discretionary Access Terms

Table 21-2 describes the terms used with discretionary access to objects:

Table 21-2: Discretionary Access Terms

Term	Definition
Read access	Either the subject is granted read permission by the object's mode and ACL, or the subject is root.
Write access	Either the subject is granted write permission by the object's mode and ACL, or the subject is root.
Execute access	Either the subject is granted execute permission by the object's mode and ACL, or the subject is root.
Search access	Either the subject is granted search permission by the directory's mode or the subject is root.
Owner access	Either the subject's EUID is the same as the object's UID, or the subject is root.

21.4 ACL Data Representations

Two internal ACL data structures are defined by the POSIX specification. In addition, there is a textual representation for ACLs that is presented to the user. You must be careful not to mix data structures when using the ACL system calls and library routines.

All data representations use the following basic types, most of which are

required by the POSIX ACL specification:

```
typedef unsigned short  acl_tag_t;
typedef unsigned short  acl_permset_t;

/*      acl_tag_t values        */

#define BOGUS_OBJ               0
#define USER_OBJ                1
#define USER                    3
#define GROUP_OBJ               4
#define GROUP                   5
#define OTHER_OBJ               6

/*      acl_permset_t values    */

#define ACL_NOPERM              0x0
#define ACL_PEXECUTE            0x1
#define ACL_PWRITE              0x2
#define ACL_PREAD               0x4

/*      tag qualifier type      */

typedef union {
        uid_t   _acl_uid;
        gid_t   _acl_gid;
} acl_id;
```

21.4.1 Working Storage Representation

Many of the ACL routines operate on 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, your 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. The working storage representation of an ACL can be converted to other representations of an ACL, as described in Section 21.4.2.

21.4.2 Data Package Representation

The data package represents an ACL in a contiguous section of memory that is independent of the process address space in which it resides. The data package can be passed between processes or stored in a file because it is contiguous and does not have internal pointer references.

The data structure associated with the data package is visible at the programmer interface. It contains a header that includes the number of entries in the ACL and a magic number to identify the data structure type, followed by an array of ACL entries that contain the tag type, permissions, and qualifier.

The following structure describes the header:

```
/*
 *      Header for data package type.
 */

typedef struct {
        ushort  acl_num;
        ushort  acl_magic;
} aclh_t;
```

In addition to including the header, the data structure for the data package includes an array of ACL entries of the following format:

```
/*
 *      Descriptor for a specific ACL entry
 *      in internal representation.
 */
typedef struct {
        acl_tag_t       acl_tag;
        acl_permset_t   acl_perm;
        acl_id          acl_id;
} acle_t;
/*
 *      Descriptor for specific ACL entry
 *      in (data package format).
 */
struct acl_data {
       aclh_t       acl_hdr;
/*     acle_t       *acl_entry;  comment; acl entries follow */
};
typedef struct acl_data         *acl_data_t;
```

The following code fragment operates on all entries in a data package ACL representation:

```
        acl_data_t     data_p;
        acle_t         *entry_p;
        int            i;

        entry_p = (acle_t *) (data_p + 1);
                  /* The pointer arithmetic addresses the first
                     acl entry.*/
        for (i = 0; i < data_p->acl_num; i++, entry_p++) {
                /* reference the ACL entries through entry_p */
        }
```

21.4.3 External Representation

The external representation is a textual, human readable version of the ACL that corresponds to the text package described in the POSIX specification and the `acl`(4) reference page. The external representation is composed of a sequence of lines, each of which is terminated by a newline character. Table 21-3 shows how the individual entries are structured.

Table 21-3: 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

The external representation is used by the POSIX routines that convert between the working storage representation and the text package.

21.5 Default ACLs

The POSIX specification allows a default access ACL to be associated with directories. When a directory has a default access ACL, files and directories created in the directory are protected with DAC attributes that are derived from the default ACL and the mode that is specified on the object creation system call. You can set a default access ACL on a directory by using the `acl_write()` call with a second argument of `ACL_TYPE_DEFAULT`. You can retrieve a default ACL by using the `acl_read()` call with a second argument of `ACL_TYPE_DEFAULT`.

Note that, although the system consults the owner and group of the directory when it is checking for access against the access ACL, the owner and group are not inherited from the default access ACL when a file is created in a directory that has a default access ACL. The default access ACL does not include an owner and group ID. Rather, the base user and group entries include permissions that are used when computing the access ACL for a file; the file owner is set from the process and the group is inherited from the parent directory.

The Digital UNIX system provides an extension of the POSIX ACLs in the form of an added default directory ACL type that can be used by other ACL implementations (for example, the Distributed Computing Environment

(DCE)). This ACL is ACL_TYPE_DEFAULT_DIR and is used for directories only. If a parent directory has ACL_TYPE_DEFAULT_DIR, then new directories are set to the parents ACL_TYPE_DEFAULT_DIR.

The inheritance rules are described in detail in the acl(4) reference page and Section 5.9 of this manual.

21.6 ACL Rules

Some interactions between the ACL and the UNIX permissions are subtle, and you may end up with different permissions than you intended because of the way that ACL system calls interact with the system calls that manipulate the UNIX DAC attributes.

The following sections describe rules for programs that handle ACLs.

21.6.1 Object Creation

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. The program must now handle the possibility that the file may also be protected by an ACL. Your program should propagate the ACL in all cases where the UNIX DAC attributes would be propagated.

When the parent directory of the file to be created has a default ACL, the owner of the created file is inherited from the process and the group is inherited from the parent directory. In addition, the specified mode is used instead of the process umask. Therefore, your program must specify a proper mode on the object creation system call and must not depend on the umask to properly protect objects.

21.6.2 ACL Replication

ACLs are preserved automatically in programs that replicate permissions.

21.6.3 ACL Validity

The ACL 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
- DCE entries must be unique

21.7 ACL Creation Example

Assume that you want to set up a file's access ACL as follows:

```
user::rwx
user:june:r-x
user:sally:r-x

group::rwx
group:mktg:rwx

other::r-x
```

The following code takes the tabular form of the ACL, creates a working storage representation of the ACL, and applies it to a file.

```
struct entries {
    acl_tag_t     tag_type;
    char         *qualifier;
    acl_permset_t perms;
} table[] = {
  { USER_OBJ,  NULL,    ACL_PRDWREX },
  { USER,      "june",  ACL_PRDEX },
  { USER,      "sally", ACL_PRDEX },
  { GROUP_OBJ, NULL,    ACL_PRDWREX },
  { GROUP,     "mktg",  ACL_PRDWREX },
  { OTHER_OBJ, NULL,    ACL_PRDEX }
};

#define TABLE_ENTRIES (sizeof(table)/sizeof(table[0]))

     acl_t       acl_p;
     acl_entry_t entry_p;
     int         i;
     uid_t       uid;
     gid_t       gid;

     /*  allocate an ACL */
     acl_alloc(&acl_p);                          1

     /* walk through the table and create entries */
     for (i = 0; i < TABLE_ENTRIES; i++) {

          /* allocate the entry */
          acl_create_entry(acl_p, &entry_p);     2

          /* set the permissions */
          acl_set_perm(entry_p, table[i].perms);

          /* setting the tag type and qualifier depends
             on the type */
          switch (table[i].tag_type) {

          case USER:
```

```
            /* map user name to ID and specify as qualifier */

            uid = pw_nametoid(table[i].qualifier);  3
            acl_set_tag(entry_p, table[i].tag_type,
                        (void *) uid);              4

            break;

         case GROUP:

            /* map group name to ID and specify as qualifier */

               gid = gr_nametoid(table[i].qualifier);  5
               acl_set_tag(entry_p, table[i].tag_type,
                           (void *) gid);
               break;


         default:

               /* qualifier is NULL for other types */

               acl_set_tag(entry_p, table[i].tag_type, NULL);
               break;
         }
}


/* set the ACL on the file */

if (acl_write(filename, ACL_TYPE_ACCESS, acl_p) < 0)
        perror(filename);

/* free storage allocated for the ACL */

acl_free(acl_p);
```

Notes:

1 This demonstrates the use of the allocation call for a working storage representation of the ACL.

2 A new ACL entry is allocated with this call. The tag type, qualifier, and permissions have an unspecified type.

3 The `pw_nametoid()` routine is an optimized mapping from user name to ID. It is described in the `map_ids()` reference page.

4 The `acl_set_tag()` function takes a `void` argument for the qualifier, and casts it to the appropriate data type, depending on `obj_type`.

5 The `gr_nametoid()` routine is an optimized mapping from group name to ID. It is described in the `map_ids()` reference page.

21.8 Imported and Exported Data

When imported and exported objects with ACLs are accessed, several issues arise that vary depending on the scenario.

21.8.1 Digital UNIX System to Same Digital UNIX System

From a Digital UNIX system to the same Digital UNIX system, the ACL does not have to be converted from the internal to the external format. It can be saved in its compacted data format. This saves the effort of having to convert the ACL from the external to internal format and back.

21.8.2 Digital UNIX System to Another Digital UNIX System

If the two Digital UNIX systems have the exact same password and group files (that is, these files are distributed using NIS or some other means), then saving the ACL is done the same as from a Digital UNIX system to the same Digital UNIX system. If the systems do not share the same password and group files, then the ACLs must be converted to the external format when backing up. If this is not done, unintended access may be granted to the files being restored. Also, this ensures that all the UIDs and GIDs are known.

21.8.3 Digital UNIX System to Other

If you are exporting data to another machine, the remote system's ACL policy needs to be based on Draft 13 of POSIX P1003.6 (or something compatible).

21.8.4 Other to Digital UNIX System

When importing data containing ACLs from another vendor's machine, the ACL format must be based on Draft 13 of POSIX P1003.6. The ACL is validated and then set on the object if valid. If the ACL cannot be set on the object being restored, 0700 is added to the standard UNIX permissions.

File Summary A

Table A-1 contains a summary of all the files that are in the trusted computing base (TCB) on the trusted Digital 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

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/gr_id_map	Binary group name to ID map
/etc/auth/system/pw_id_map	Binary user name to ID map
/etc/auth/system/subsystems	Printable names for protected subsystems
/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

Table A-1: (continued)

File Name	Remarks
/etc/passwd	Accounts database
/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
/sbin/sh	Shell
/sbin/sulogin	Single-user root login password verifier
/sbin/swapon	Add swap devices
/sbin/umount	Unmount mounted file systems
/tcb/bin/XIsso	ISSO role program
/tcb/bin/XSysadmin	System administrator role program
/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/files/auth/<a-z>/*username*	Protected password file
/tcb/files/auth.db	Protected password database for system accounts

Table A-1: (continued)

File Name	Remarks
/var/tcb/files/auth.db	Protected password database for user accounts
/tmp	Temporary directory
/users	Parent of users home directory
/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
/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(2) system call

Table A-1: (continued)

File Name	Remarks
/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 /etc/passwd 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
/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
/usr/lib/libsecurity.a	Security-relevant library routines

Auditable Events and Aliases B

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 Digital 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  fail
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
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
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
getdirentries        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
audcntl                 succeed  fail
setsysinfo              succeed  fail
swapctl                 succeed  fail
memcntl                 succeed  fail

SystemV/unlink          succeed  fail
SystemV/open            succeed  fail
RT/rt_setprio           succeed  fail

!  Audited trusted events:
audit_start             succeed  fail
audit_stop              succeed  fail
audit_setup             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
```

B.2 Sample Event Aliases File

The following is the sample `/etc/sec/event_aliases` file provided

with the Digital 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 \
            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:     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
```

Interoperating with and Migrating from ULTRIX Systems C

This appendix describes some of the issues you may encounter when moving applications and accounts from an ULTRIX system to a Digital UNIX system.

C.1 Migration Issues

The following sections describe migration issues you may encounter when moving from ULTRIX to Digital UNIX.

C.1.1 Difference in the audgen System Call

Applications built under ULTRIX, that make use of the `audgen()` system call, do not work on Digital UNIX because the Digital UNIX version of `audgen()` takes five parameters instead of three as on ULTRIX. To port these applications, you can take either of the following steps:

- Convert ULTRIX-style usage of `audgen()` to the OSF-style usage. For example:

  ```
  /* ULTRIX */
  audgen(event, tokenmask, param_vector);
  ```

  becomes:

  ```
  /*Digital UNIX*/
  audgen(event, tokenmask, param_vector, NULL,NULL);
  ```

- Link such applications with the following module:

  ```
  #include <sys/syscall.h>
  #include <stdio.h>
  audgen(event, tokenp, argp)
  int event;
  char *tokenp;
  char *argp[];
  {
     return(syscall(SYS_audgen, event, tokenp, argp, \
                                                NULL, NULL));
  }
  ```

C.1.2 Differences in the audcntl Routine

The Digital UNIX `audcntl()` routine takes six parameters instead of five as on ULTRIX. You need to put a zero (0) in the unused parameter.

C.1.3 Changes to the authaudit Routines

If you are moving from ULTRIX MLS+ or a system based on OSF code, several of the audit routines in the code base have been superceded in the Digital UNIX operating system by the `audgen()` and `audgenl()` routines. The routines are provided only for backward compatibility and will be removed in a future release. The routines are:

```
audit_security_failure()
audit_no_resource()
audit_auth_entry()
audit_subsystem()
audit_login()
audit_rcmd()
audit_passwd()
audit_lock()
sa_audit_lock()
sa_audit_audit()
```

The functions of the `audit_adjust_mask()` routine have been superceded by `audcntl()`.

See Chapter 19 for examples of how to use the `audcntl()` and `audgenl()` routines. More information on `audgen()`, `audgenl()`, and `audcntl()` is available in the associated reference pages and the `auduit.h` file.

C.1.4 Difference in the Authentication Interfaces

The Digital UNIX SIA authentication interfaces are different from the ULTRIX interfaces.

C.1.5 Differences in Password Encryption

The Digital UNIX system uses a form of password encryption that is different from that used on ULTRIX. An ULTRIX system has three security levels: BSD, UPGRADE, and ENHANCED. A Digital UNIX has only two security levels: BASE (equivalent to BSD) and ENHANCED. There is not a direct equivalent to the ULTRIX UPGRADE security level. There are only direct equivalents to BSD and ENHANCED modes. This is because the default Digital UNIX ENHANCED password encryption algorithm is compatible with the traditional password encryption, which is not the case for ULTRIX ENHANCED security.

Running the Digital UNIX `secsetup` script leaves the system equivalent to the ULTRIX UPGRADE level; the old password can be used once. The `secauthmigrate` script uses the ULTRIX ENHANCED password encryption algorithm, which is not compatible with the traditional style password encryption algorithm. If `secauthmigrate` is going to be used, run the `secsetup` script before running `secauthmigrate`.

C.1.6 Trusted Path Unavailable on Digital UNIX

The ULTRIX trusted path feature is not available on Digital UNIX systems.

C.1.7 Secure Attention Key (SAK) Unavailable on Digital UNIX

The ULTRIX secure attention key (SAK) feature is not available on Digital UNIX systems.

C.2 Moving ULTRIX Authentication Files to Digital UNIX

Users whose records are being transferred must have valid BSD style login records (with the exception of valid password fields) on the ULTRIX system. This can be through NIS as well as a local record in `/etc/passwd`. (This is checked with the `ls –o`.) You might want to do an account review, so that only those users who should still have active accounts are moved.

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

C.2.1 Converting Shared Authentication Files

Use the following procedure to convert ULTRIX shared authentication files (BIND/Hesiod) to DEC OSF/1 authentication files:

1. On the ULTRIX system, make a copy of the distributed authentication data as follows:

   ```
   # cp –p /var/dss/namedb/src/auth /tmp/auth.hesiod
   ```

2. Copy the `/tmp/auth.hesiod` file to the Digital UNIX system.
3. If the BSD style profile information for the ULTRIX systems is shared by NIS, it is necessary to copy the `/var/dss/namedb/src/passwd` file to the Digital UNIX system. Add this file to the NIS password maps or append it to the `/etc/passwd` file.

4. Run the /usr/sbin/secauthmigrate script as follows:

   # **/usr/sbin/secauthmigrate auth.hesiod**

   You should test the script by setting the ROOTDIR environment variable to a temporary location as follows:

   # **/usr/bin/env ROOTDIR=/tmp /usr/sbin/secauthmigrate auth.hesiod**

5. Continue the migration by going to Section C.2.3.

C.2.2 Converting Local Authentication Files

Use the following procedure to convert the ULTRIX files:

1. Because the /etc/auth file is not normally up-to-date, use getauth to obtain the current values from /etc/auth.{pag,dir} as follows:

   # **umask 077**
   # **getauth > /tmp/auth.local**

2. Copy the /tmp/auth.local file to the Digital UNIX system.
3. Run the /usr/sbin/secauthmigrate script as follows:

   # **/usr/sbin/secauthmigrate auth.local**

   You should test the script using the ROOTDIR environment variable first.
4. Continue the migration by going to Section C.2.3.

C.2.3 After Converting the Authentication Files

If any accounts are left in /tcb/files/auth/?/*user*:ULT, it is because there was already a protected profile for the *user*. Use the following procedure to complete the migration:

1. Merge the values as appropriate. Edit the file using a duplicate, copy the new file to /tcb/files/auth/?/*user*:t,
2. Check to be sure that the base file (/tcb/files/auth/?/*user*) has not been changed. If it has, merge the change into the /tcb/files/auth/?/*user*:t file.
3. Rename the /tcb/files/auth/?/*user*:t file to /tcb/files/auth/?/*user*.

If a UID is not known, the secauthmigrate script reports that it cannot

translate a UID to a name using the following code:

```
# ls –o /tmp/file
```

This test is performed on a file owned by the UID in question. It may be necessary to check the contents of `/etc/passwd` or the NIS setup. If this discrepancy persists, it indicates that there was an orphaned authentication record in the original ULTRIX data.

Once all the records have been converted, review their contents with the `dxaccounts` program.

C.3 Audit Data Compatibility

The following are compatibility issues between the auditing subsystems on ULTRIX and Digital UNIX systems:

- Audit data on a Digital UNIX system is not compatible with audit data on an ULTRIX system.
- Audit data generated on an ULTRIX system is read using the `audit_tool.ultrix` program. See the `audit_tool`(8) reference page for more information.
- The Digital UNIX `auditd` and the ULTRIX `auditd` do not communicate with each other.
- The `auditd` command line is different between ULTRIX and Digital UNIX systems. See the `auditd`(8) reference page for details.
- The `auditd` access control list, which was found in `/etc/auditd_clients` on ULTRIX, is found in `/etc/sec/auditd_clients` on Digital UNIX systems.

Coding Examples D

The examples in this appendix illustrate how to use some of the routines in the trusted Digital UNIX system.

D.1 Source Code for sia-reauth.c

Example D-1 is a program that performs password checking.

Example D-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) {
```

Example D-1: (continued)

```
            (void) sia_ses_release(&entity);
            sleep(3);   /* slow down attacks */
            (void) fprintf(stderr, "sorry");
            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;
}
```

D.2 Source Code for sia-suauth.c

Example D-2 is a program that allows root to become a user to run daemons (such as `crontab` or `sendmail`) for the user.

Example D-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);
```

Example D-2: (continued)

```
        printf("SIA authentication for uid: %d, uname: %s ", \
                                                        uid, uname);
        if (sia_ses_init(&entity,oargc,oargv,NULL,uname,NULL, \
                                    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);
}
```

Symbol Preemption for SIA Routines E

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.

E.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.

E.2 The Digital UNIX Solution

To satisfy both ANSI C and developer expectations, Digital 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()`.

Note

In this online version of the book, the long underscore (__) in front of the function name is actually two underscore characters (_ _).

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.

E.3 Replacing the Single-User Environment

Example E-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 E-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.

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.

ACL (access control list)
: An optional extension of the traditional UNIX permission bits, which gives the user the ability to specify read/write/execute permissions on a per user basis.

Access ACL
: The formal name of the ACL that is checked for access decisions on an object.

AIC (Attribute IR Cache)
: A cache mechanism for storing the ACL IR in memory to eliminate duplicate ACLs and reduce the number disk accesses.

auditing
: The recording, examining, and reviewing of security-related activities on a trusted system.

audit event
: An event that is monitored and reported on 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)
: ID that is created at login time and that is inherited across all processes.

BASE security
: The traditional security that is delivered on BSD UNIX systems, BASE security consists of file permissions. A nontrusted Digital UNIX system has BASE security.

BSD (Berkeley Software Distribution)
: UNIX software release of the Computer System Research Group of the University of California at Berkeley –- the basis for some features of the Digital UNIX and ULTRIX operating system.

Default ACL
: An ACL that is associated with directories. This type of ACL determines the Access ACL of any file created in that directory. New directories inherit the default ACL from the parent directory as both the Access and Default ACL.

discretionary access control (DAC)
: The traditional UNIX form of file permissions set with the `chmod` command.

entity
: SIA introduces the term entity to mean a user, program, or system which 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 or interchanges between foreign systems.

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 security
: The optional security features that supplements BASE security. Enhanced security consists of extended password profiles and the audit subsystem.

entity
: Term used by the security intergration architecure to define a user, program, or system that can be authenticated.

evaluation criteria
: The *Trusted Computer System Evaluation Criteria (TCSEC)*. The enhanced security features in the Digital UNIX system have been designed to meet this criteria.

IR (internal representation)
: A binary representation of an ACL, that can be easily converted into the Distributed Computing Environment (DCE) ACL binary format.

ISSO (information system security officer)
: In a trusted system, the person traditionally responsible for ensuring the security of the system. The person who serves this administrative role is your contact for all security-related questions. The ISSO sets up an initial authentication profile, which specifies login restrictions and passwords options.

  The ISSO is also responsible for auditing system activity, setting the security characteristics of devices, and performing other security-related tasks. See also **system administrator**.

login spoofing program
: Any program that represents itself as a `login` program in order 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.

object (as defined for ACLs)
: An object as defined for an access control list, refers to the following data storage entities:

  - File system entries
  - Semaphores
  - Message queues
  - Shared memory

operator
: The person responsible for the day-to-day maintenance of a system, including backups, line printer maintenance, and other routine maintenance tasks.

Privileged process
: A process that can bypass the permission checks for an operation. If privileges are not configured in the system, then the process must be running with the effective id of 0 (root). If privileges are configured, the process must possess the appropriate granular privilege.

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 the programmer has explicitly designed it to uphold the security policies of the system. See also **process**.

PPID, 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.

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
: In the trusted system, the person responsible for administrative tasks that are not performed by the ISSO. The system administrator is responsible for file system maintenance and repair, account creation, and other miscellaneous administrative duties. In many cases, the system administrator acts as a balance of power to the ISSO. See also **ISSO**

TCB (trusted computing base)
: The set of hardware, software, and firmware that together enforce the system's security policy. The Digital UNIX TCB includes the system hardware and firmware as delivered from Digital, the trusted Digital UNIX operating system, and the trusted commands and utilities that enforce the security policy. The operating system and all of the other software distributed with the trusted Digital UNIX system have been modified to 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

M

N

R

S

T

U

V

W

X

How to Order Additional Documentation

Technical Support

If you need help deciding which documentation best meets your needs, call 800-DIGITAL (800-344-4825) before placing your electronic, telephone, or direct mail order.

Electronic Orders

To place an order at the Electronic Store, dial 800-234-1998 using a 1200- or 2400-bps modem from anywhere in the USA, Canada, or Puerto Rico. If you need assistance using the Electronic Store, call 800-DIGITAL (800-344-4825).

Telephone and Direct Mail Orders

Your Location	Call	Contact
Continental USA, Alaska, or Hawaii	800-DIGITAL	Digital Equipment Corporation P.O. Box CS2008 Nashua, New Hampshire 03061
Puerto Rico	809-754-7575	Local Digital subsidiary
Canada	800-267-6215	Digital Equipment of Canada Attn: DECdirect Operations KAO2/2 P.O. Box 13000 100 Herzberg Road Kanata, Ontario, Canada K2K 2A6
International	————	Local Digital subsidiary or approved distributor
Internal[a]	————	SSB Order Processing – NQO/V19 *or* U. S. Software Supply Business Digital Equipment Corporation 10 Cotton Road Nashua, NH 03063-1260

[a] For internal orders, you must submit an Internal Software Order Form (EN-01740-07).

Reader's Comments

Digital UNIX
Security
AA-Q0R2D-TE

Digital welcomes your comments and suggestions on this manual. Your input will help us to write documentation that meets your needs. Please send your suggestions using one of the following methods:

- This postage-paid form
- Internet electronic mail: `readers_comment@zk3.dec.com`
- Fax: (603) 881-0120, Attn: UEG Publications, ZKO3-3/Y32

If you are not using this form, please be sure you include the name of the document, the page number, and the product name and version.

Please rate this manual:	Excellent	Good	Fair	Poor
Accuracy (software works as manual says)				
Completeness (enough information)				
Clarity (easy to understand)				
Organization (structure of subject matter)				
Figures (useful)				
Examples (useful)				
Index (ability to find topic)				
Usability (ability to access information quickly)				

Please list errors you have found in this manual:

Page	Description

Additional comments or suggestions to improve this manual:

What version of the software described by this manual are you using?

Name/Title Dept.

Company Date

Mailing Address

Email Phone

Do Not Cut or Tear − Fold Here and Tape

digital™

NO POSTAGE
NECESSARY IF
MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST−CLASS MAIL PERMIT NO. 33 MAYNARD MA

POSTAGE WILL BE PAID BY ADDRESSEE

DIGITAL EQUIPMENT CORPORATION
UEG PUBLICATIONS MANAGER
ZKO3−3/Y32
110 SPIT BROOK RD
NASHUA NH 03062−9987

Do Not Cut or Tear − Fold Here

Cut on
Dotted
Line