ULTRIX

digital

Security Guide for Users

Order Number: AA–PBKQA–TE

ULTRIX

Security Guide for Users

Order Number: AA-PBKQA-TE
June, 1990

Product: ULTRIX Version 4.0 or higher

digital equipment corporation
maynard, massachusetts

Contents

About This Manual

1 Understanding Your Role in Maintaining Security

2 Protecting Your Account

3 Protecting Your Files and Directories

4 Processes and Shells

5 Connecting to Other Systems

6 Workstation and Windowing Environments

Examples

Figures

Tables

About This Manual

The *Security Guide for Users* describes the security features available in the ULTRIX-32 operating system. Some of these security features may not be available at a particular site. Whether or not a security feature is available at your site depends upon how your site has configured the ULTRIX environment to meet its security needs.

Audience

This manual is intended for users who have some experience in an ULTRIX environment. Users should understand basic ULTRIX commands, know how to set file permissions, and understand basic shell syntax.

The *Security Guide for Users* describes security concepts whose understanding is needed to help create a secure computing environment. The guide often provides examples of ways to implement security concepts. However, the guide is not intended as a tutorial for beginning ULTRIX users. Readers should be familiar with the material in *The Little Gray Book: An ULTRIX Primer*, and be able to apply security concepts in their own environments.

This guide is not intended for users who program. Users who need general guidelines for writing secure programs in the ULTRIX environment should read the *ULTRIX Languages and Programming Guide.*

Organization

The *ULTRIX Security Guide for Users* has six chapters, a glossary, a security summary, and an index.

Chapter 1 **Understanding Your Role in Maintaining Security**

Introduces the notion of computer security, describes some common security threats, and describes the roles that general users and the security administrator play in maintaining system security.

Chapter 2 **Protecting Your Account**

Shows how the system verifies your identity and how you can check for recent logins to your account. Tells how you can protect your account by logging in through a trusted path, and which login messages should be reported to your security administrator. Provides ideas for creating secure passwords. Explains how to leave your terminal or workstation during the day and how to log out when leaving.

Chapter 3 **Protecting Your Files and Directories**

Provides a brief description of ULTRIX file-protection mechanisms and reviews how you can set permissions for your files. Describes the interaction between some system commands and file permissions.

Chapter 4 **Processes and Shells**

Defines a process and discusses the difference between real and effective user IDs (UIDs) and group IDs (GIDs). Shows how SUID/SGID (set UID and set GID) programs affect the privileges accorded a process. Discusses how to create a secure environment for the login shell process.

Chapter 5 **Connecting to Other Systems**

Discusses the security concerns for commands that log in to, execute commands on, and copy files from remote systems.

Chapter 6 **Workstations and Windowing Environments**

Provides an overview of the special security concerns inherent in workstation and windowing environments.

Appendix A **Glossary**

Defines ULTRIX security terms.

Appendix B **Security Summaries**

Gathers together the security summaries from Chapters 2 through 6.

Related Documents

DECnet-ULTRIX User's and Programmer's Guide

DECwindows User's Guide

The Little Gray Book: An ULTRIX Primer

ULTRIX Guide to System Environment Setup

ULTRIX Languages and Programming Guide

ULTRIX Reference Pages

ULTRIX Security Guide for Administrators

Conventions

The following conventions are used in this manual:

hostname>	The default user prompt is your system name followed by a right angle bracket. In this manual, a percent sign (%) is used to represent this prompt.
#	A number sign is the default superuser prompt.
UPPERCASE lowercase	The ULTRIX system differentiates between lowercase and uppercase characters. Literal strings that appear in text, examples, syntax descriptions, and function definitions must be typed exactly as shown.
macro	In text, bold type is used to introduce new terms.
`user input`	This bold typeface is used in interactive examples to indicate typed user input.
`system output`	This typeface is used in interactive examples to indicate system output and also in code examples and other screen displays. In text, this typeface is used to indicate the exact name of a command, option, partition, pathname, directory, or file.
rlogin	In syntax descriptions and function definitions, this typeface is used to indicate terms that you must type exactly as shown.
filename	In examples, syntax descriptions, and function definitions, italics are used to indicate variable values; and in text, to give references to other documents.
[ ]	In syntax descriptions and function definitions, brackets indicate items that are optional.
`cat`(1)	Cross-references to the *ULTRIX Reference Pages* include the appropriate section number in parentheses. For example, a reference to `cat`(1) indicates that you can find the material on the `cat` command in Section 1 of the reference pages.
CTRL/*x*	This symbol is used in examples to indicate that you must hold down the CTRL key while pressing the key *x* that follows the slash. When you use this key combination, the system sometimes echoes the resulting character, using a circumflex (^) to represent the CTRL key (for example, ^C for CTRL/C). Sometimes the sequence is not echoed.
RETURN	This symbol is used in examples to indicate that you must press the named key on the keyboard.

Understanding Your Role in Maintaining Security 1

This chapter provides a brief introduction to the importance of computer security. It discusses the roles that users and security administrators play in creating and maintaining a reasonable level of security in an ULTRIX computing environment.

Access to information in computer systems is a balance between openness and secrecy. On one hand, users need to share information with relative ease. On the other hand, security administrators and users must guard against having information copied or removed by unauthorized personnel. Neither the operating system nor the security administrator can stop you from giving away information that you own. However, you should be aware of the security tools at your disposal and the risks engendered if you use them improperly.

As you read this guide, remember that computer security is not a static discipline; each major technological advance in computing seems to spawn a new set of security issues. Designing secure computers is an evolving discipline. Although some security standards exist (for example, the *Department of Defense Trusted Computer System Evaluation Criteria*, DoD 5200.28-STD, December 1985), the tools and techniques employed by those who design computers and those who try to break into them are constantly changing.

1.1 What Is Computer Security?

When you leave home, you lock the doors to decrease the chances that someone might break in and destroy or steal your possessions. You could install burglar alarms, balancing the inconvenience of using these security devices against any potential loss.

In a computing environment, maintaining security is more difficult. For example, you may not know that someone has broken into your account; if someone copies your files, nothing is missing, but the information is no longer secret. Locking the doors does not guarantee that the information is safe.

When discussing computer security, two terms are frequently used: **physical security** and **information security**.

- **Physical security** is protecting the computer itself from physical attack; for example, locking a computer room to prevent the theft or destruction of hardware or backup tapes provides a measure of physical security.
- **Information security** is protecting the programs and information contained within the computer from attack; for example, setting restrictive file protections on a sensitive file provides a measure of information security.

Before networks and modems were commonplace, physical security provided a degree of information security. Although physical security will always be a security consideration, it no longer provides adequate information security. When a computer is connected to a network or a modem, security risks increase exponentially because an attacker no longer needs physical access to the computer, or to any terminals

connected directly to it. Probably the only truly secure computer is a single-user, fault-tolerant system locked in a bank vault with no network or phone access — and no power.

The emphasis in this guide is on information security because the security features available to users in the ULTRIX operating system help protect information. Information security has four aspects: **secrecy**, **integrity**, **availability**, and **accountability**.

- **Secrecy** is protecting information from unauthorized disclosure. Password-protected accounts allow only valid users to log in to a system. File permissions limit the rights of other users to read your files.
- **Integrity** is protecting information from unauthorized modification. File permissions limit the rights of other users to modify or delete your files.
- **Availability** is protecting the system from denial-of-service attacks that range from misusing system resources to crashing the system. Your security administrator has programs to monitor system resource use.
- **Accountability** is identifying and holding a user accountable for actions that create, modify, provide access to, or disseminate information. Your security administrator has programs capable of tracking the actions of an individual user.

1.2 Threats to Information Security

Information security for ULTRIX systems depends on the notion of the authorized user; anyone who knows the user name and password for an account can log in and be granted all the privileges associated with that account. For this reason, an attacker will often try to gain access to a user account, with the goal of eventually gaining access to the `root` account and its superuser privileges.

Two common techniques used by attackers to steal information are the masquerade and Trojan horse programs. These two techniques represent deliberate attempts by other users or attackers to breach security.

A more pervasive but less spectacular threat is an authorized user who does not employ available security features to protect accounts and files adequately. This is akin to protecting your house by nailing the doors shut and leaving the windows open. In other words, the best security features in the world are worthless if not used.

1.2.1 Masquerade Programs

A **masquerade program** pretends to be a legitimate part of the operating system. This program then attempts to fool you into revealing sensitive information.

A common masquerading technique is a program called a **password grabber**. This program runs at an unattended terminal and waits for a user to attempt a login. Masquerading as the system's login process, the program prompts for a user name and password, which it steals and copies to a location specified by the attacker. After stealing the user name and password, the program displays a false error message and returns control to the real operating system. If you are caught by this type of program, you will probably assume that you have incorrectly entered your password, not knowing that you have just given it away.

1.2.2 Trojan Horse Programs

A **Trojan horse program** gains access to sensitive information by pretending to serve a valid purpose, while concealing its real intent. Almost any program can be turned into a Trojan horse by inserting a hidden subprogram into it. While the program performs its advertised function, the subprogram takes advantage of the user's privileges to copy, modify, or execute any files that the user can access.

For example, a Trojan horse could be concealed in a spelling-checker program. While the program checks your spelling, it secretly copies files from your account to the attacker's account or to a public account.

Note

The success of the masquerade and Trojan horse techniques depends on the attacker's ability to plant or modify a program on your system. An attacker does not need to gain access to a user account if a user copies a program of this type into an account and runs it. For this reason, never copy a file to your system and execute it unless you know what is in it.

1.3 Overview of ULTRIX Security Features

The ULTRIX system is a multitasking, multiuser system. It can perform more than one task at a time and share resources among many users. Precisely because it is a multiuser system, it has to perform a number of tasks that directly or indirectly affect security. The following list describes some of these tasks:

- The system maintains separate user accounts, checking user names and passwords at login and putting users in the correct accounts. This ensures that only authorized users can log in, which affects information secrecy, integrity, and availability.
- The system assigns each user a process at login time, keeps track of any processes created by this process, and ensures that resources allocated to any process are available for the life of the process. The system also records all login attempts and command usage. These records provide accountability.
- The system checks ownership and permissions each time a process attempts to access a file to ensure that the process has permission to read, write, or execute the file. This file-access control provides information secrecy and integrity.

On an ULTRIX system, users assign permissions to their files and control, to a large extent, the security of their accounts. This provides both freedom and flexibility, but demands that users behave in a responsible manner and provide the degree of security required by their environment. The following analogy may help clarify some system security issues.

Your user account on an ULTRIX system is similar to that of a tenant's apartment in an apartment building. The front entrance to the building is controlled by the landlord who passes out keys to tenants. You can decorate your own apartment. You can also give other tenants permission to look at, change, and play with any of your possessions. However, you cannot look in other tenants' apartments without permission.

The landlord's apartment has a lot of expensive furniture, plus the keys for all the other apartments. Usually you are allowed to look around this apartment and play with certain items; however, this is entirely at the

discretion of the landlord. You are not allowed to remove or change anything important in the landlord's apartment.

If a burglar breaks into your apartment, the following might happen:

- Your possessions can be stolen or ruined (`mv` or `rm` commands).
- The burglar can find out who else has apartments (by looking at the `/etc/passwd` file).
- A cunning burglar might take pictures of (`cp`) your secrets, hide a spy camera (perhaps some type of Trojan horse), and leave without a trace.
- The burglar may be able to break into the landlord's apartment and put the whole building at risk by getting the keys to all the other apartments.

The risks are even greater if the burglar is one of the tenants.

1.4 User and Security Administrator Security Roles

Users often greet security measures with disdain, assuming that most security measures are restrictive, annoying, and a waste of their time. Oftentimes this is because users do not understand the risks involved in using computers as a tool for creating, storing, and sharing sensitive information. Because the ULTRIX system gives users some control over the security of their accounts, both users and security administrators share responsibility for watching over the system.

1.4.1 User

Users must protect their accounts and files. As a user, you should know the system well enough to take reasonable precautions when performing an operation that could affect security. You should report any signs of a security breach to your security administrator. For example, if you know you have typed your password correctly, but your login attempt fails.

1.4.2 Security Administrator

The security administrator implements security policy by controlling logins and access to system-level files and services. The security administrator is responsible for auditing the system to check for security breaches. On a well-managed system, the security administrator makes sure that users understand the security policy and the reasons behind it. The security administrator might also be responsible for the physical security of the equipment and the configuration of certain aspects of the file system that have security implications.

Note

On many systems, the system manager also acts as the security administrator. However, because the role of security administrator differs from that of system manager, this guide refers to the person responsible for security matters as the security administrator.

The following chapters explore ULTRIX security issues and provide the information you need to make rational decisions about security in your own account.

Protecting Your Account 2

To a large extent, security in an ULTRIX environment depends on denying an attacker entry to a system. Once in a system, the odds favor an attacker because even extremely restrictive file permissions offer no protection if the attacker becomes the owner of the files. Therefore, protecting your account is your first line of defense.

This chapter describes:

- How the system identifies authorized users and allows them to log in
- How to invoke a trusted path for a secure login
- Which login messages and system files can help you determine if someone has logged in to your account
- How to choose good passwords and protect them
- What to do when you leave your terminal
- How to ensure that logout procedures have executed

2.1 How the System Knows Who You Are

The system knows who you are by your account entry in the `/etc/passwd` file. Your entry has your login name, your password (if your security administrator is storing it there), information about your account, and two numbers that identify you to the system: your user ID (UID) and your group ID (GID). The system uses your UID and GID to control your actions and your access to files. For more information on `/etc/passwd`, see `passwd`(5) in the *ULTRIX Reference Pages.*

When you log in, the system prompts for your user name. It then prompts for your password. Because only you should know your password, the system accepts the correct entry of the password as valid proof that you are really you. Most systems have users prove who they are only at login. Therefore, from the system's point of view, anyone who knows your user name and password is you.

Whenever someone attempts to log in, the system requires both user name and password information before allowing or denying access. The system could be programmed to display a `login incorrect` message whenever someone entered an invalid user name, and not prompt for a password. But if the system did not display a password prompt when someone entered an invalid user name, an attacker could use this information to distinguish between valid and invalid user names.

After the system verifies your identity at login, it creates a process for you and assigns the UID and GID from your `/etc/passwd` entry to this process, along with any additional GIDs from the `/etc/group` file. You have only one UID, but can belong to more than one group. Groups are discussed in greater detail in Chapter 4. For more information on `/etc/group`, see `group`(5) in the *ULTRIX Reference Pages.*

Whenever you issue a command or run a program, the system starts a process for you and assigns your UID and GIDs to that process. This enables the system to keep track of what you are doing and to determine whether or not you can open a file or execute a command. Each time you ask the system to perform an action, it checks the rights associated with your process.

2.2 Logging In

This guide assumes that you know how to log in to your ULTRIX account. If you do not, refer to *The Little Gray Book: An ULTRIX Primer* for an introduction to ULTRIX systems.

Note

Because displays may differ for terminals and workstations, the messages and system output shown in examples throughout this guide might not match those displayed on your screen. However, the principles behind the examples are similar. See Chapter 6 in this manual for an overview of security concerns relating to workstation and windowing environments.

2.2.1 Invoking a Trusted Path

When you log in from a hardwired or Local Area Transport (LAT) terminal, the program displaying the first `login:` prompt is `getty()`. This program calls `login()`, which displays the `Password:` prompt and continues the login sequence.

A classic masquerade tactic is to steal user names and passwords by imitating the `getty()` and `login()` programs. To defeat this tactic, the ULTRIX operating system provides a **trusted path** to `getty()`. This trusted path ensures that you are communicating with the real `getty()` program and not with a masquerade program. When you invoke a trusted path, the system terminates all processes attached to your terminal (thus terminating any masquerade program) and then starts `getty()`. This ensures that your user name and password are passed only to the correct system programs.

Your security administrator can tell you if the trusted path mechanism is enabled on your system. If it is, the security administrator has defined a key that you must press to invoke a trusted path. This key is called the Secure Attention Key (SAK).

Note

There is no "SAK" key on your keyboard. The system administrator defines one of the existing keys as the Secure Attention Key. If the BREAK key is defined as your Secure Attention Key and your terminal connects to a terminal server, there is a potential conflict. Most terminal servers intercept a BREAK keystroke and interpret it as a request to display the terminal server prompt. In this case, your request for a trusted path will always be intercepted and interpreted by the terminal server. To resolve this conflict, simply change the terminal server attention key to some other key. For example, to change the terminal server attention key on a DECserver 200 from BREAK to CTRL/A type the following at the `LOCAL>` prompt:

```
LOCAL> set port local CTRL/A
```

2.2.2 Checking Login Messages

At login, pay attention to the messages that indicate both your last login time and any recent login failures. For example:

```
login: hale
password:      <not displayed>
Last login: Fri May 12 07:26:53 on tty17
There have been 0 unsuccessful login attempts on your account
```

- If the last login time displayed is not the last time you logged in, tell your security administrator.
- If any reported login failures were not your own, tell your security administrator.

Do not ignore any evidence that someone might have attempted to get into your account.

2.2.3 Understanding Login Auditing

The system maintains a record of recent logins and logouts in `/usr/adm/wtmp`. To display a list of recent logins for your account, use the `last` command with your user name as the argument. For example:

```
% last hale

hale     tty16            Mon May 15 07:39   still logged in
hale     tty17            Fri May 12 07:26 - 16:58  (09:32)
hale     tty16            Thu May 11 07:23 - 12:14  (04:51)
hale     tty16            Wed May 10 07:36 - 16:57  (09:20)
hale     tty20            Tue May  9 09:32 - 16:53  (07:20)
hale     tty16            Mon May  8 12:20 - 17:36  (05:16)
hale     tty17            Mon May  8 07:38 - 10:23  (02:44)
```

If you cannot remember any recorded logins, tell your security administrator. By checking your login messages for failed login attempts and using the `last` command to check for unauthorized logins, you should be able to spot both failed and successful attacker login attempts.

Your security administrator also keeps an eye on failed login attempts. The system maintains records of failed logins in the `/usr/spool/mqueue/syslog*` files. You have read access to these files. Five successive failed logins generate a warning message in the current system log. This alerts your security administrator to a possible attack.

In addition to tracking failed login attempts, your security administrator can record all invocations of the login program. The security administrator can selectively retrieve information and check for abnormal events. If there is a security breach, the security administrator can reconstruct the details of the attack.

2.3 Passwords

Password security is one of the cornerstones of system security. Your security administrator selects your first password when creating your account. Because security risks increase whenever more than one person knows the password for an account, always change your password immediately upon receiving a new account.

When you change your password, the password you enter is encrypted and stored. Whenever you log in and enter your password, your entry is encrypted and checked against your stored password. If they match, you are allowed to log in.

Note

Encryption is the process of transforming data to an unintelligible form in such a way that the original data either cannot be obtained (one-way encryption) or cannot be obtained without using the inverse decryption process (two-way encryption). The ULTRIX operating system uses one-way encryption for passwords.

2.3.1 Where Passwords Are Stored

Historically, encrypted passwords and user account information are stored as entries in the `/etc/passwd` file. Although `root` owns `/etc/passwd`, anyone on the system can read this file. This generally presents no problem because the password is encrypted. However, a sophisticated attacker could obtain a copy of `/etc/passwd`, encrypt commonly used passwords, and look for matches to valid passwords.

The ULTRIX Version 4.0 operating system provides the option of storing encrypted passwords in `/etc/passwd` or in a data base file called `/etc/auth`. Read and write access to `/etc/auth` is restricted to `root`. Your security administrator determines whether or not encrypted passwords are stored in `/etc/passwd` or in `/etc/auth`.

2.3.2 Maximum and Minimum Password Lifetimes

If your password is stored in `/etc/auth`, your security administrator can increase security by setting some limits on your password.

- Setting a maximum and a minimum password lifetime.

  A maximum lifetime forces you to change your password on a regular basis by causing your password to expire after a specified length of time. A minimum lifetime specifies a length of time that you must keep a password before you can change it. This helps you resist the temptation of changing to a new password and then changing back to your old familiar one.

  If your password has a maximum lifetime, the system displays a login message starting five days before your password expires. The message tells you the number of days remaining before your password is invalid. This reminds you to change your password before it expires. To display your password expiration date at any time, use the `shexp` command.

  ```
  % shexp
  Expires Tue Dec  6 10:49:18 EST 1989
  ```

- Setting a minimum password length.

  When using `/etc/auth`, a password can be up to 16 characters in length. Your security administrator can specify a minimum password length. This increases security because the longer the password, the larger the pool of possible choices, and the harder it is for an attacker to guess an individual password.

2.3.3 Keeping Your Password Secret

Protecting your account depends on choosing a good password, keeping it secret, and changing it regularly. Use the following suggestions to keep your password secret:

- Select a password that is hard to guess.
- Make sure that no one is watching when you enter your password; some people can remember keystrokes. Also, you could type your password when you meant to type your user name; your password would be echoed on the screen.
- Keep your password or passwords in your head, not in a file, on a piece of paper, or mapped to a function key on an intelligent terminal. Do not include your password in an electronic mail message.
- Change your password regularly.
- If you have more than one account, use a different password for each account. Otherwise, an attacker who breaks into one account has a key that opens more than one door.

2.3.4 Avoiding Bad Passwords

If you are allowed to choose your own passwords, avoid choosing ones that are easily guessed. After reading the following list, you should be able to tell if your current password or passwords offer much resistance to an attacker running a password guessing program.

Do not use the following as passwords:

- Your user name
- Human and pet names
- Words found in dictionaries and spell-checking programs
- Street and city names
- Well-known fictional characters and places
- Any of the previous items spelled backwards
- License plate and social security numbers
- Passwords under six characters in length

2.3.5 Choosing Good Passwords

Good passwords take advantage of the variety of upper- and lower-case characters available on the keyboard. You can use characters like the underscore (_) or square bracket ([) to make your password more complex. For example, **Try_this[1** is an acceptable password.

Use the following suggestions when creating a password:

- Make sure that your passwords are six or more characters in length.
- Use **passphrases** (passwords created by stringing words or pronounceable syllables together; for example, **nottoworry**).
- Create an acronym from an uncommon phrase (for example, the phrase "I want to live in New Zealand" could become **iw2liNZ**).

- Use a mix of alphabetic and nonalphabetic characters (for example, **not.2.worry**).
- Use a mix of upper- and lower-case characters (for example, **nOt2worry**).

The best passwords combine several of these suggestions, as in the case of **nOt_2.woRry**. However, do not create a password that is so complex you cannot remember it.

2.3.6 Using System-Generated Passwords

On most systems, you can choose your own password. However, your security administrator can specify that the `passwd` command display a list of acceptable passwords. Each password is divided into syllables, which are separated by hyphens to make it easier to pronounce and remember. This hyphenated form of the password appears next to each password. If you do not like any of the passwords on the first list displayed, you can display other lists of acceptable passwords. However, you do not have the option of creating your own password.

If you have the option of choosing your own password, you can still have the system provide a list of acceptable passwords by issuing the following command:

```
% passwd -a
```

Note that passwords generated by `passwd` meet the guidelines set forth in the *Department of Defense Password Management Guideline*, CSC-STD-002-85.

Note

If your system uses Yellow Pages (YP), use the `yppasswd` command to change your password.

2.3.7 Dealing with Expired Passwords

In addition to setting a maximum lifetime for your password, your security administrator can set a **soft expiration period** for your account. This option gives you one, and only one, chance to log in with an expired password.

If your account has a soft expiration period and you log in with an expired password, the `passwd` command is automatically executed. You must change your password before the login procedure will complete and put you in your home directory.

If you do not have a soft expiration period and your password expires, you will have to ask your security administrator to create a new password so you can log in.

Workstations

If you use a workstation and your password is due to expire within the next five days, the workstation displays a list of system-generated passwords. This list is similar to the list you can generate with the **passwd -a** command. You can click on the MORE button to display additional system-generated passwords.

You can type in your own new password, or you can choose one of the system-generated passwords.

2.4 Leaving Your Terminal

Train yourself to take the following two steps whenever you leave your terminal, no matter how short a time you plan to be away from it. (If you use a workstation, also see Chapter 6.)

1. Return to a shell prompt (usually % or $) and enter the `clear` command to clear the screen.
2. If you are connected to a terminal server that allows you to lock the port for your terminal, you have two options:
   a. Lock the terminal server port. Digital's terminal servers such as the DS100, DS200, DS300, DS500/500 and ETS support the `lock` command, which prompts for a password and then locks the port. To unlock the port, enter the password.
   b. Lock your current ULTRIX session by typing `lock` at the shell prompt. The system prompts you for a password and then locks the session. Note that if you have other sessions that are not locked, anyone can use your keyboard to access the terminal server and resume a session you forgot to lock. To unlock the session, enter the password. For more information on the `lock` command, see `lock`(1) in the *ULTRIX Reference Pages.*

Note

Because you deny access to any sessions connected to that port, locking the port is the preferred method.

2.5 Logging Out

When you log out, make sure that you have really logged out. It is easy to press the wrong keys when you are in a hurry, and leave a live session behind you. Most users have at one time or another come to work to find a live session from the previous day.

Also, if you have stopped jobs, the system might not log you out the first time you try to log out. For example, the C shell displays the following message if you have any stopped jobs and attempt to log out.

```
There are stopped jobs.
```

You have the choice of checking on the jobs and then logging out or reissuing the logout command. If you forget that you have stopped jobs in the background, you might issue one logout command and walk away. You think you have logged out, but the system is waiting for you to respond to its message.

If the terminal is in a public area, the security risk is increased. Take the time to verify that you have indeed logged out of your current session by checking for a logout verification message on your screen.

If your terminal is connected to a terminal server, verify that you have logged out of all sessions connected to the port. Digital's terminal servers such as the DS100, DS200, DS300, DS500/500 and ETS support the `show sessions` command, which lists any active sessions on the port.

If you have responsibility for the physical security of your terminal or workstation, make sure that any physical locks on the equipment or on the room containing the equipment are securely locked before leaving.

2.6 Security Summary

Your account is your first line of defense:

- If enabled on your system, use the Secure Attention Key to ensure that your user name and password are passed only to the correct system programs at login. This defeats attempts to steal this information.
- Pay attention to login messages that contradict your memory of your last login attempts. Use the `last` command to check recent logins to your account.
- Protect your password:
  - Pick a good one. Consider using a passphrase or a system-generated password.
  - Keep it secret. Do not write it down. Do not let others see you type it.
  - Change it regularly.
  - Use different passwords for different accounts.
- Lock your session whenever leaving your terminal during work hours.
- Verify that you have terminated all sessions before leaving work.

Protecting Your Files and Directories 3

This discussion of file and directory protection assumes that you have some knowledge of ULTRIX file permissions, including the use of the `chmod` command and the `umask` shell command. For a basic introduction to file permissions, refer to *The Little Gray Book: An ULTRIX Primer.*

This chapter tells you how to:

- Display file permissions with the `ls -lg` command and interpret the output.
- Provide the minimum file permissions required for your work by setting a restrictive file creation mask with the `umask` command and modifying permissions as needed with the `chmod` command.
- Control file access for groups of users with the `chgrp` command.
- Create separate directories for files with similar security requirements.
- Check for both suspicious files and poorly protected files with the `find` command.
- Understand how file permissions and ownership affect command execution, and how some commands affect file permissions and ownership.
- Encrypt and decrypt sensitive information.

3.1 File Types, Ownership, and the ls -lg Command

This section discusses the file types recognized by the ULTRIX operating system, the rules that determine the owner and group associated with a file, and the output from the `ls -lg` command.

3.1.1 File Types

The ULTRIX file system recognizes three types of files:

Ordinary Files whose structure and contents can be controlled by users. Text files and executable files are examples of ordinary files.

Directory Files that contain information about other files. The structure of a directory file is controlled by the system. You can create a directory using the `mkdir` command, but the system controls how the information is stored in the file.

Special Files that control system input/output (I/O). There are special files for terminals, disk and tape drives, printers, and so forth. Links, sockets, and pipes are also examples of special files.

You are primarily responsible for protecting your ordinary files and directories. The special files that control system I/O to storage and output devices are generally created and owned by `root`.

3.1.2 File Ownership

The ULTRIX operating system uses the ownership and permission information associated with each file to determine which users can read, write, or execute the file.

When a file is created, the system assigns the effective UID from the process creating the file to the file. The user name associated with the UID becomes the owner of the file. The system assigns the GID from the directory containing the file to the file. The group name associated with this GID becomes the group associated with the file.

UID Determines who owns the file. Files that you create are assigned your UID. Only `root` can create a file with a UID other than its own UID. The owner of the file has *user* access to the file.

GID Determines which group has potential access to the file (actual access is determined by the *group* permissions associated with the file). A process that tries to open a file and does not have a UID or GID that matches the file is automatically placed in the category of *other*.

Because file ownership is important for file system maintenance and accounting purposes, only the superuser can execute the `chown` command to change the owner of a file. Some system commands automatically change ownership, for example the `cp` command, but users cannot give away their files or take another's.

You can change the group associated with a file or directory you own. The `chgrp` command is discussed in this chapter.

3.1.3 The ls -lg Command

The `l` and `g` options to the `ls` command provide a long listing that includes the name of the group associated with each file. The output from the `ls -lg` command for an ordinary file looks like this:

```
% ls -lg text
-rw-r--r--  1  hale   staff   6744  May 26  17:01    text
```

The following list refers to this example when describing the output format:

- File Type (-):
  The first character of the long listing is the file type. The following characters are used to indicate file types:

-	ordinary file	**l**	symbolic link (special file)
d	directory file	**p**	pipe (special file)
c	character-type special file	**s**	socket (special file)
b	block-type special file		

- File Permission (`rw-r--r--`):
  The file permission consists of nine characters, in three groups of three. Each group indicate the read, write, and execute permissions granted to *user*, *group*, and *other*. A hyphen (-) in the permission area indicates that the permission in question is denied. In the previous example, read and write access is given to *user*, read access to *group*, and read access to *other*.

- Link Count (`1`):
  The number following the file permission is the number of directory entries that directly reference this file. Tables 3-5 and 3-6 provide information on the security implications associated with links.

- File Owner (`hale`):
  The user name from `/etc/password` that matches the UID associated with the file. You should own the files in your account. If files owned by other users are in your account, you should know why they are there.
- Group (`staff`):
  The group name from `/etc/group` that matches the GID associated with the file. One method that uses group names to control access to files is discussed later in this chapter.
- Size (`6744`):
  The size of the file in bytes.
- Modification Time (`May 26 17:01`):
  The date when the file was last modified. Both file size and modification time provide security-relevant information. If you cannot account for any changes in size or modification time, tell your security administrator. This chapter provides a shell script to help you check for potential security problems relating to file permissions.
- File Name (`text`):
  The name of the file.

For more information on the `ls` command, see `ls`(1) in the ULTRIX Reference Pages.

3.2 Maintaining Restrictive File Permissions

You own the primary responsibility for protecting your files and directories. Only you or the superuser can change the permissions on your files. A good guideline for file permissions is to restrict access to your files and directories to the minimum needed to perform your work. You can always add permissions on a file-by-file basis.

The `umask` and `chmod` commands are two tools that you use to create and maintain secure file permissions. The `umask` command sets an environment variable that affects files created in the future; `chmod` changes permissions on existing files. You should use `umask` to create files with restrictive permissions and use `chmod` to modify these permissions as needed.

Note

The `umask` command is both a shell built-in command and a system call. Although there is no reference page for `umask` in section 1 of the *ULTRIX Reference Pages*, this guide refers to `umask` as a command to avoid confusion.

3.2.1 Using Octal Numbers and Symbolic Values for File Permissions

File permissions can be expressed either as octal numbers or as symbolic values. The `chmod` command accepts both formats, the `umask` command accepts only octal numbers. You should be able to specify and interpret permissions in either format. Table 3-1 shows how read, write, and execute permissions are expressed in both formats.

Table 3-1: Octal Numbers and Symbolic Values

Permission	Octal Number	Symbolic Value
execute	1	x
write	2	w
read	4	r
read + execute	5	rx
read + write	6	rw
read + write + execute	7	rwx

When specifying a file permission in the octal format, read, write, and execute permissions are represented by three octal numbers, one each for *user*, *group*, and *other*. Each number represents the sum of granted permissions. For example, the number **6** indicates read (**4**) and write (**2**) permission. A number's position indicates whether the permissions in question are granted to *user*, *group*, or *other*. For example, an octal permission of **751** grants the following permissions:

- Read, write, and execute to *user*
- Read and execute to *group*
- Execute to *other*

When specifying a file permission in the symbolic format, read, write and execute permissions are represented by the characters **r**, **w**, and **x**. The characters **u**, **g**, **o**, and **a** determine whether the permission in question is granted to *user*, *group*, *other*, or *all*. The **=**, **+**, and **-** operators determine whether the new permissions replace, add to, or subtract from the current permissions. For example, a symbolic permission of **u=rwx, g=rx, o=x** grants the following permissions:

- Read, write, and execute to *user*
- Read and execute to *group*
- Execute to *other*

Example 3-1 shows how the same file permission can be specified using octal numbers or symbolic values.

Example 3-1: Octal and Symbolic File Permissions

```
% chmod 751 test.file_1

% ls -l test.file_1
-rwxr-x--x 1  hale        15944  Apr 17  17:35    test.file_1

% chmod u=rwx,g=rx,o=x test.file_2

% ls -l test.file_2
-rwxr-x--x 1  hale        10189  Apr 28  09:25    test.file_2
```

Note

In addition to read, write, and execute permissions, there are three other permissions you can assign to files: set user ID (SUID), set group ID (SGID), and the sticky bit. However, because most work with file permissions involves read, write, and execute permissions, the sections on the `umask` and `chmod` commands focus only on these permissions. Table 3-4 includes the octal and symbolic representations for all six permissions. SUID and SGID files have known security implications; these permissions and implications are discussed in Chapter 4. The sticky bit is discussed later in this chapter.

3.2.2 Setting Your File Creation Mask with the umask Command

Your file creation mask determines the file permissions assigned to any files you create. There is no standard default file permission; however, many programs set the default permission to **666** for ordinary files and **777** for directory and executable files. Your *file creation mask* removes permissions from these default permissions, thus determining the actual permissions assigned to a new file.

The `umask` command sets or displays your file creation mask. Therefore, understanding and using the `umask` command properly ensures that your files are created with a secure set of permissions.

For example, a `umask` of **027** removes write permission for *group* and removes read, write, and execute permissions for *other*. This would change a default permission of **666** on a text file to **640**, and a default permission of **777** on a directory or executable file to **750**.

Example 3-2 shows how a `umask` value of **027** modifies the default permission assigned to a text file. (Although `umask` requires an octal number as its argument, the example shows permissions in both octal and symbolic format for clarity.)

Example 3-2: How the File Creation Mask Determines File Permissions

```
                                  octal                    symbolic
                                u   g   o                  u   g   o

default file permission (666)   420 420 420 (666)      rw- rw- rw-
umask (027)                     000 020 421 (027)      --- -w- rwx
                                -----------            -----------
resulting permission            420 400 000 (640)      rw- r-- ---
```

When you provide an octal argument, `umask` sets your file creation mask. Without an argument, `umask` displays the current file creation mask. For example:

```
% umask 027                <sets the umask value to 027>
% umask                    <displays the current umask value>
027
```

In addition to using `umask` as a shell command, you should set a `umask` value in your shell startup files. It is important to set a reasonably restrictive `umask` value in your `.profile` file for the Bourne shell, or in the `.login` file for the C shell. How restrictive a `umask` value you choose depends on the level of security you decide best fits your work.

Setting your umask to **027** is a good starting point for file protection. You grant read, write, and execute permission to yourself; you allow members of your group to read and execute your files (but they cannot modify or delete anything); and you deny any access to all others.

Table 3-2 shows examples of umask values and the resulting permissions for text files and executable or directory files.

Table 3-2: Examples of umask Values

umask Value	Description	Text File	Executable or Directory File
002	no write for other	rw-rw-r	rwxrwxr-x
006	no read or write for other	rw-rw----	rwxrwx--x
007	no access for other	rw-rw----	rwxrwx---
022	no write for group and other	rw-r--r--	rwxr-xr-x
026	no write for group, no read or write for other	rw-r-----	rwxr-x--x
027	no write for group, no access for other	rw-r-----	rwxr-x---
077	no access for group and other	rw-------	rwx------

3.2.3 Changing File Permissions with the chmod Command

You can change the permission on any file you own with the chmod command.

The chmod command accepts both octal numbers and symbolic values as arguments. Octal arguments are useful when you want to set **absolute permissions** on one or more files. That is, no matter what the existing permissions are, you want to replace them with the specified permissions. Example 3-3 shows how to set an absolute permission on a file.

Example 3-3: Setting Absolute Permissions with the chmod Command

```
% chmod 700 test.file_1

To check the permission, type the following:

% ls -l test.file_1
-rwx------  1  hale        12089  Apr 30  09:55    test.file
```

No matter what the permissions on the file were, the file now grants read, write, and execute access to *user*, and no access to *group* or *other*.

A symbolic argument to chmod assigns an absolute permission (using the = operator) or a relative permission (using the + or – operators). A **relative permission** adds to or subtracts from the existing files permissions. This is useful when you have files with different permissions in the same directory, and want to change the same permissions on all files, without affecting other permissions.

Using the = operator with no arguments removes all permissions for that category. For example, **o=** removes all permissions for *other*.

Example 3-4 shows the symbolic assignment of relative permissions to a file.

Example 3-4: Setting Relative Permissions with the chmod Command

```
% ls -l test.file
-rwx------  1  hale        12089  Apr 30  09:55    test.file
% chmod g+rx,o+x test.file
To check the permission, type the following:
% ls -l test.file
-rwxr-x--x  1  hale        12089  Apr 30  09:55    test.file
```

There is one instance where the `chmod` command interacts with the `umask` value. If you do not specify **u**, **g**, **o** or **a**, the new permission is automatically applied to *user*, *group*, and *other*. However, your `umask` value now affects the assigned permission, as it does when you create a new file. Example 3-5 shows how a `umask` value of 027 affects the resulting file permissions in this case.

Example 3-5: Interaction Between the umask and chmod Commands

```
% ls -l date.file
-rwxr--r--  1  hale            12089  Apr 30  09:55    date.file
% chmod +wx date.file
% ls -l date.file
-rwxr-xr--  1  hale            12089  Apr 30  09:55    date.file
```

The `umask` value denied write permission for *group*, and denied write and execute permission for *other*. Note that the `umask` value did not remove read permission for *other*; it only prevented the addition of write and execute permission.

3.2.4 File Permission Command Summary

Table 3-3 lists some common commands that either change file permissions or the owner or group associated with a file.

Table 3-3: File Permission Command Summary

Task	Command	Argument	Example
Set file creation mask	`umask`	octal	`% umask 027`
Show file creation mask	`umask`	none	`% umask`
Set absolute file permission	`chmod`	octal	`% chmod 750` *filename*
		=	`% chmod u=rwx,g=rx,o=` *filename*
Set relative file permission	`chmod`	+ or -	`% chmod u+w,o-rwx` *filename*
Display file permissions	`ls`	`-lg`	`% ls -lg`
Change owner	`chown`	n/a	(must be superuser)
Change group	`chgrp`	group name	`% chgrp` *group_name filename*

3.2.5 File Permission Reference Table

Table 3-4 summarizes file permissions and their effect on files and directories. The table includes octal values for permissions, and shows the position required when using octal values as arguments to the `umask` or `chmod` commands.

Table 3-4: File Permission Reference Table

Symbolic	Octal	File	Directory
r	0444	Read permission.	Permission to list (`ls`) the directory.
w	0222	Write permission.	Permission to add or remove files.
x	0111	Execute permission.	Permission to change to a directory (`cd`) or use the directory in a search path.
s	4000 2000	Set user ID (SUID). Set group ID (SGID). Sets the effective UID or GID of the process executing a file to that of the owner or group of the file, depending on whether the "s" is in the *user* or *group* field.	n/a n/a
t	1000	Sets the **sticky** (save text image) **bit**. Tells the operating system to keep an executable file in the swap area and not overwrite it. Only the superuser can set this bit on a nondirectory file.	When the sticky bit is set on a directory, only the owners of files in the directory (or the superuser) can delete the files. For example, the sticky bit is normally set on `/tmp`, a directory owned by `root` which contains temporary files owned by different users.

3.3 Using Groups and Directories to Control Access to Files

In addition to using file permissions to control access to files, you can also control access through careful use of the group mechanism and by creating directories for similar types of files.

3.3.1 Changing the Group Associated with a File

If you own a file, you can use the `chgrp` command to change the group associated with it. You must be a member of the group you want to associate with the file. Example 3-6 shows how the `groups` command displays the names of groups in `/etc/group` that list you as a member.

Example 3-6: Using the groups Command

```
% groups
staff sec_pol security rev_board
```

Example 3-7 shows how to use the `chgrp` command to change the group associated with a file.

Example 3-7: Using the chgrp Command on a File

```
% ls -lg test.file
-r--r--r--  1  hale   staff     12089  Apr 30  09:55    test.file

% chgrp security test.file

% ls -lg test.file
-r--r--r--  1  hale   security  12089  Apr 30  09:55    test.file
```

When you change the group associated with an ordinary file, you affect only that file, but when you change the group for a directory, any files created in that directory from now on are automatically associated with the new group.

Example 3-8 shows how to increase security by using the `chgrp` command on the directory used to store files for a specific project.

Example 3-8: Using the chgrp Command on a Directory

Assume that you work on a project and want to maintain a common directory for project files. This directory will be in your account. You want to allow members of the project group to read, write and execute files in this directory, but you do not want to give them *group* access to the rest of the files in your account.

1. Ask your security administrator to create a new group in `/etc/group` that contains only the members of this project. (The name of this group in the example is `project_A`.)
2. Create a directory for the group:

   ```
   % mkdir project_dir
   ```

3. Change the group name associated with the directory to `project_A`:

   ```
   % chgrp project_A project_dir
   ```

4. Set the desired permissions on the directory and check the results:

   ```
   % chmod 770 project_dir

   % ls -ldg project_dir
   drwxrwx---  2  hale  project_A   512  May 30  011:35   project_dir
   ```

Note

If the members of the group want to copy files into the directory, make sure that both the directory and the files in it have write permission set for *group*. Otherwise, although the directory grants write access, the system will not let group members overwrite a file that does not grant *group* write permission. Of course, because the members of the group have write access to the directory, they could always remove any existing files and then copy in the newer versions, but it is easier to set write permission for *group* on existing files.

To restrict the removal of files from the project directory to the owners of the files, set the sticky bit on the project directory. As mentioned in Table 3-4, when the sticky bit is set on a directory, only the owner of a file, the owner of the directory, or the superuser can remove a file from the directory. The next example shows how to set the sticky bit on a directory:

```
% chmod 1770 project_dir

% ls -ldg project_dir
drwxrwx--t  2  hale  project_A   512  May 30  011:45   project_dir
```

This command sets the sticky bit on the directory, grants read, write, and execute permission to *user* and *group*, and denies all access to *other*.

3.3.2 Using Directories to Increase Security

You cannot set `umask` values on a directory-by-directory basis. The `umask` value you select applies to all files created by your process. For this reason, it is a good idea to create directories for files with similar security requirements. You can then use the `chmod` command with the wildcard character (*) to set permissions for an entire directory, rather than having to set permissions on a file-by-file basis.

3.4 Checking File Permissions and Ownership with the find Command

If you are like most users, you eventually create numerous directories and more files than you can remember. It is a time-consuming task to check each directory looking for suspicious files.

To check your directories for file permissions that may need attention, use the following shell script. A **shell script** is an executable text file that contains a series of shell commands. This script also checks `/tmp` for any files owned by you in that directory. If you use the script, substitute your name for *username*, make the script executable, and restrict access so that only you can modify it.

```
#! /bin/sh
# This script checks file permissions and ownership on files residing in
# your account. It also looks for files owned by you in /tmp

echo "   Files with Write Access for Other"
find $HOME -perm -002 -exec ls -ld {} ;

echo "   Files with Write Access for Group"
find $HOME -perm -020 -exec ls -ld {} ;

echo "   Files with Changes in the Last Seven Days"
```

```
find $HOME -mtime -7 -exec ls -ld {} ;

echo "   SUID Files"
find $HOME -perm -4000 -exec ls -ld {} ;

echo "   SGID Files"
find $HOME -perm -2000 -exec ls -ld {} ;

echo "   Files Not Owned by User"
find $HOME ! -user username -exec ls -ld {} ;

echo "   Files in /tmp"
find /tmp -user username -exec ls -ld {} ;
```

Because this script could produce a long list of files, you might want to redirect the script output to a file rather than having it display only on the screen.

3.5 How File Permissions Affect Command Execution

Permissions directly affect execution of the following commands:

- `ar`, `cpio`, and `tar`
- `cat`, `more`, `less`, `head`, and `tail`
- `cd`
- `cp`
- `ln`
- `ls`
- `mv`
- `rm`
- `rmdir`

In some instances, permissions inhibit command execution. For example, in order to use the `cat` command to display a file on your screen, you must have execute permission on the directory the file is in and read permission on the file.

Table 3-5 summarizes the permissions required for successful execution of several commands. Note that you must have execute permission for each directory in the pathname of a file, regardless of whether the file is a source file or a destination file.

Table 3-5: File Permissions Required for Successful Command Execution

Command	Required Permissions
`ar, cpio, tar`	Must have read permission on the source file and execute permission on the source directory. If the destination file does not exist, must have write and execute permission on the destination directory. If the destination file does exist, must have write permission on the file and execute permission on the directory. Note that some options to these commands might affect the required permissions.

Table 3-5: (continued)

Command	Required Permissions
`cat`	Must have read permission on the file and execute permission on the directory the file is in. These permissions are also the minimum required for other commands that display a file, such as `more`, `less`, `head`, or `tail`.
`cd`	Must have execute permission on the directory you want to change to.
`cp`	Same as `ar`.
`ln`	Hard link. A link is a directory entry that either points directly to information about an existing file (a hard link) or points to another directory entry for an existing file (a symbolic link). For more information about links, see `ln`(1) in the *ULTRIX Reference Pages*. For hard links, must have write and execute permission on the directory the hard link is in and execute permission on the directory the file is in.
`ln -s`	Symbolic links. Must have write and execute permission on the directory the symbolic link is in, do not need any permissions on the directory the file is in. In fact, you can create a symbolic link to a nonexistent file. For both hard and symbolic links, the permissions on the file determine access, not the permissions on the link.
`ls`	Must have read permission on the directory.
`ls -l`	Must have read and execute permission on the directory.
`mv`	Must have write and execute permission on the source directory. Must have write and execute permission on the destination directory. If moving a directory to another directory on the same file system, must have write permission on the directory you are moving plus write and execute permission on its parent directory. If the source and destination files reside on different file systems, `mv` does not rename the file, but performs a `cp` and an `rm` of the file.
`rm`	Must have write and execute permission on the directory containing the file. If there is no write permission on the file, the system prompts for confirmation before removing the file. If the sticky bit is set on the parent directory, must be the owner of the file, the owner of the directory, or the superuser.
`rmdir`	Must have write and execute permission on the parent directory. Do not need any permissions on the directory to be removed, which must be empty.

3.6 How File-Manipulation Commands Affect File Permissions

To reduce security risks, the system automatically changes the permission and ownership associated with a file when the following commands are executed:

- `cp`
- `cpio`
- `ln`
- `mv`
- `tar`

Table 3-6 summarizes the actions taken by the system in these instances.

Table 3-6: File Permissions Affected by Successful Command Execution

Command	Affected Permissions
`cp`	If you copy a file and the destination file does not exist, the source file permissions are copied with the file. However, if the source file is owned by `root` and has any SUID, SGID, or sticky bits set, these bits are not set on the destination file. Your default `umask` affects the file permissions, as it does when you create any file. If you are not the owner of the source file, you become the owner of the destination file. If you copy a file to a directory owned by another user, you are still the owner. If the destination file exists, the source file overwrites it (only if write access is allowed), but the permissions are not affected.
`cpio`	If you use `cpio` to copy files from an archive, you become the owner of the files. Only the superuser can copy files into the system and have the files retain their original owners. Because ownership is affected when copying files into the system, physical protection of archive media is important. If you copy all your files to a tape and another user obtains the tape, that user can recreate your files and become the owner.
`ln`	You can create a hard link (`ln`) to a file only if the link and the file are on the same file system. This limitation exists because any hard links to the file, and the initial directory entry for the file, contain the inode number of the file. (The inode number identifies a structure that contains information about the file, such as its actual location, permissions, and file type. The `ls -li` command displays a long listing which includes the inode numbers for your files.)

Table 3-6: (continued)

Command	Affected Permissions
	You can create a symbolic link (`ln -s`) to a file on your file system or another file system. A symbolic link points to another directory entry, not to the file itself. The **link count** is the number found between the file permissions and the file owner in a long listing (`ls -l`). You can check the link count on a file to determine if there are any hard links to the file. You will always have at least a link count of 1 because your directory entry for the file is a hard link. A link count of 3 indicates 2 other hard links to the file. If you remove a file with hard links to it, you have only removed your hard link to the file. The other hard links can still access the file, even though you cannot. For this reason, always check the link count on a file before using the `rm` command to remove the file. If other hard links exist, strip all the permissions from the file before removing it.
`mv`	Moving a file within your own file system does not change ownership and permissions. However, when you move a file from another file system to yours, you become the owner (if you are not already) and your `umask` value affects the permissions. In addition, any links to the original file are lost.
`tar`	Similar to `cpio` command.

3.7 Using Encryption to Protect the Contents of a File

For security reasons, file encryption commands are distributed as an option in the ULTRIX V4.0 release. If the encryption commands are available on your system, you can use the `crypt` command to encrypt sensitive files. Example 3-9 shows how a short file looks before and after encryption.

Example 3-9: File Encryption

```
% cat input_file
this is an encryption test

% crypt < input_file > output_file

Enter key:  hide_me_1       <key not displayed>

% cat output_file
1X)
   5DQ'eH
         JHd
            edJ
               f{Ez
```

You could then delete the input file, leaving only the encrypted version. To retrieve the information in the encrypted file, use the `crypt` command but reverse the order of the files. Example 3-10 shows how to decrypt the file from Example 3-9.

Example 3-10: File Decryption

```
% crypt < output_file > restored_file
Enter key: hide_me_1      <key not displayed>

% cat restored_file
this is an encryption test
```

The encryption distribution also includes encryption files for use with the standard ULTRIX editors `ex`, `vi`, and `ed`. Refer to the manuals for these editors for more information.

3.8 Security Summary

Because you control file permissions on your files, understand the range of available security options, and choose those options that best fit your security needs.

- Know how the system assigns owners and groups to files.
- Set a reasonably restrictive `umask` value in your shell startup files (for example, `umask 027`).
- Provide only the minimum file permissions required for your work. You only ask for trouble when everyone can access your files. Use the `chmod` command to modify permissions as needed.
- Use the group mechanism to control file access when you need to share files with other users.
- Create separate directories for files with similar security requirements.
- Use the `find` command to check for suspicious files and files with insecure permissions.
- Understand the interaction between certain system commands (such as `mv` or `cp`) and file permissions and ownership.
- Protect removable media. If someone copies your files into the system using the `cpio` command, that user becomes the owner of your files.
- Understand how links operate on files.
- Encrypt extremely sensitive information.

Processes and Shells 4

To the ULTRIX operating system, you are defined by the files you own and the processes running on your behalf. Chapter 3, ''Protecting Your Files and Directories,'' looks at files and how to protect them. This chapter does the same for processes by:

- Providing a general description of a process.
- Explaining the differences between real and effective UIDs and GIDs.
- Discussing how SUID and SGID programs use effective UIDs and GIDs. These programs perform a valuable service, but their use without proper understanding increases security risks.
- Looking at shells and how to protect your shell startup files.

Understanding file permissions makes you aware of the security issues involved in protecting files; understanding processes increases your ability to control and monitor the security of your working environment within the system.

4.1 What Is a Process?

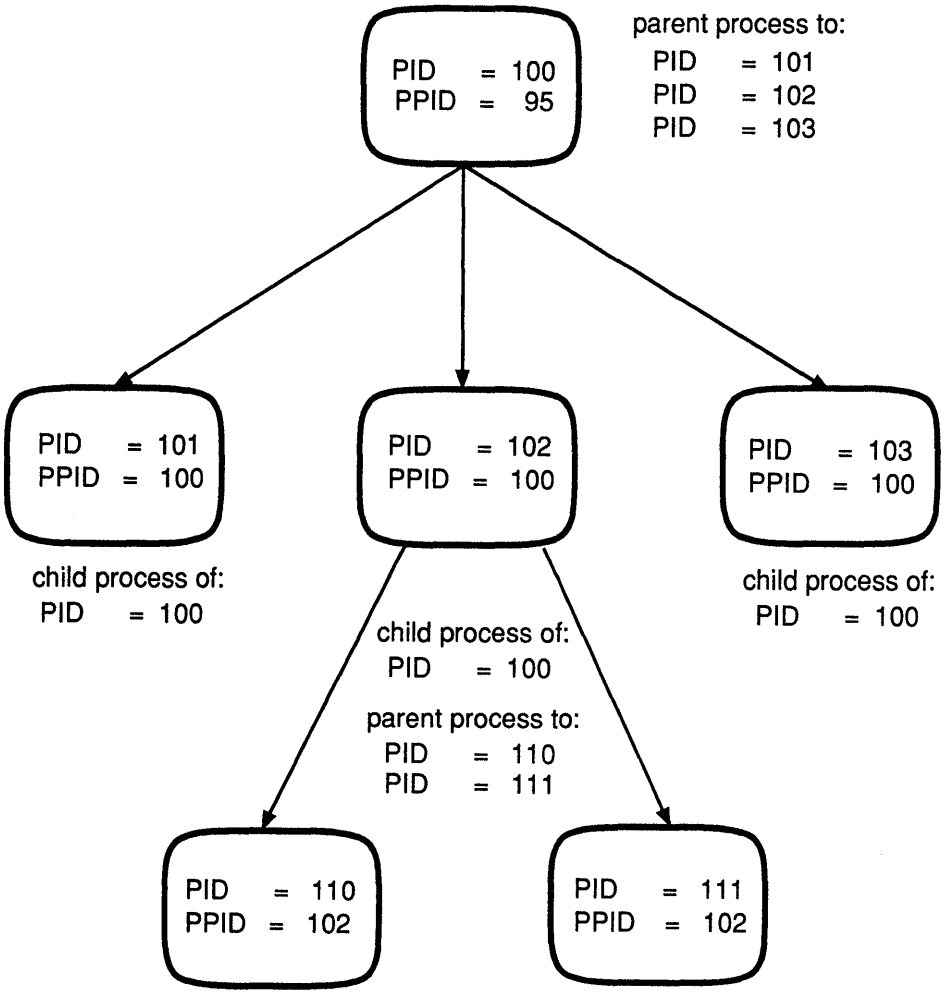
A **process** is the content (code) and the context (environment) of an executing program. Processes perform actions and accomplish tasks within the system. Some processes are created at system startup, and run until the system is shut down; for example, the `swapper`, `init`, and `pagedaemon` processes. Some processes are started when a user logs in, and run until the user logs out; for example, your login shell. Other processes exist only for the time required to perform a task. For example, when you type the `ls` command, a process is created and exists until the command finishes execution.

Processes can create other processes with the `fork()` system call.

- A process that creates another process is called the **parent process**. The created process is called a **child process**.
- A parent process can have more than one child process. However, a child process can have only one parent process.
- A child process can create its own child processes. Thus, a process can be a child of one process, yet be a parent to other processes.
- Each process has a unique identifier, called its process ID (**PID**). Each process knows its own PID and its parent's process ID (**PPID**). A child process is a copy of the parent process, except for its PID and its PPID, until it executes an `execve()` system call.
- Each process has an owner, identified by the UID associated with the process.
- A process can exit when its task is finished through the `exit()` subroutine, or can be terminated with the `kill` command or other termination signals.

Figure 4-1 shows the relationship between parent and child processes.

Figure 4-1: Parent and Child Processes



The system controls file access by comparing the UID and GIDs associated with a process to the UID and GID associated with a file. This is why you can use the `chmod` command to modify the permissions on your files, and the reason why you cannot modify the permissions on another user's files.

To display process information, use the `ps` command. You could, for example, use the `-l` and `-t` options to display a long listing of processes at the current terminal. In the following example, the name of the command executing in a process is listed under the COMMAND heading.

Example 4-1: Using the ps Command

```
% ps -lt

      F UID   PID  PPID CP PRI NI ADDR  SZ  RSS WCHAN STAT TT  TIME COMMAND
b408201 179 13824     1  3  15  031b86  36   33 fe400 S    18  0:02 sh
b008021 179 22002 13824 32  33  034064 211  194       T    18  0:00 emacs
b000001 179 22110 13824 50  37  028ea0 296  188       R    18  0:00 ps -lt
```

In the preceding example, a user with UID=179 owns the processes. From the PID and PPID information, you can see that the `sh` process (PID=13824) is the parent process of the `emacs` process (PID=22002) and the `ps` process (PID=22110). For more information on the `ps` command, see `ps`(1) in the *ULTRIX Reference Pages.*

It is a good idea to check your processes occasionally. If you find a process that you own but cannot identify, you can terminate it with the `kill` command, using the number of the unwanted process as an argument. For example:

```
% kill 22002
```

For more information on the `kill` command see `kill`(1) in the *ULTRIX Reference Pages.* Use the `ps` command to verify that the process has terminated:

```
% ps -lt

      F UID   PID  PPID CP PRI NI ADDR  SZ  RSS WCHAN STAT TT  TIME COMMAND
b408201 179 13824     1  3  15  031b86  36   33 fe400 S    18  0:02 sh
b000001 179 22122 13824 51  37  012c80 296  188       R    18  0:00 ps -lt
```

Exercise caution when terminating a process. For example, you do not want to terminate a batch job by mistake. One thing to check for would be a process owned by you but attached to a terminal you are not using. If in doubt, check with your security administrator before terminating an unknown process.

4.2 Real and Effective UIDs and GIDs

While the system assigns a UID and a GID to each file, each process has a real and an effective UID and a real and an effective GID. When determining what a process can do, the effective UID and GID are the IDs that count. They determine the actual capabilities of the process.

Real UID The UID of the owner of the process.

Effective UID The UID that determines the *user* access rights of the process. The effective UID is usually the real UID. However, in a set user ID (SUID) program, the effective UID is changed in order to modify the *user* access capabilities of the process.

Real GID The GID of the owner of the process.

Effective GID The GID that determines the *group* access rights of the process. The effective GID is usually the real GID. However, in a set group ID (SGID) program, the effective GID is changed to modify the *group* access rights of the process.

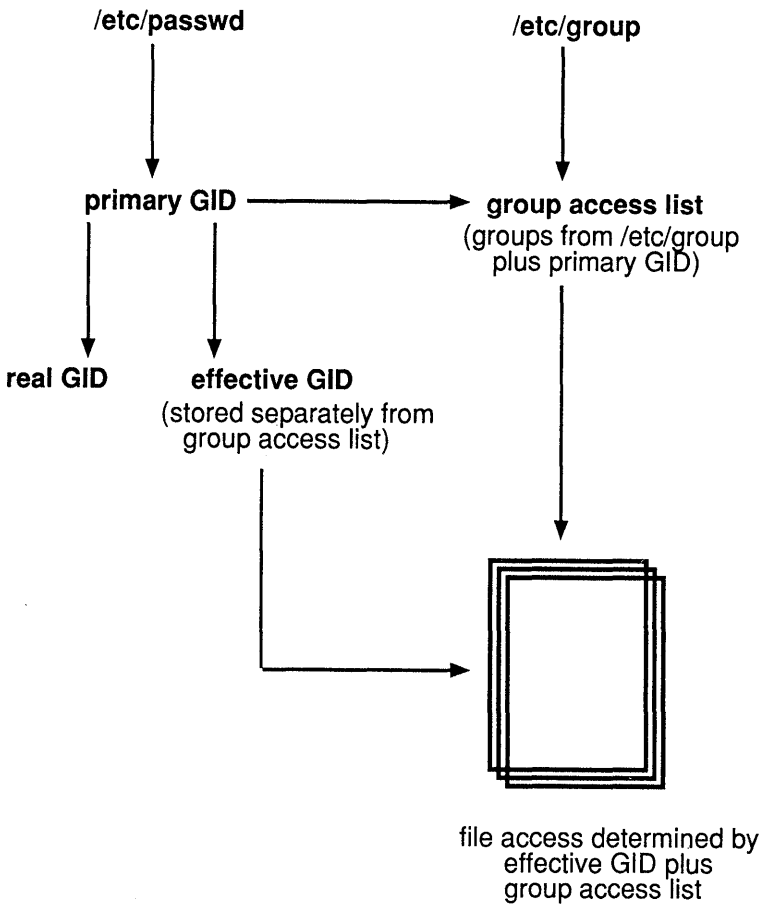
Your UID from `/etc/passwd` becomes both the real UID and the effective UID for your login shell. Any process created by your shell inherits these UIDs. Any file

you create is assigned your effective UID. Unless you execute an SUID program or run another program that changes your real or effective UID, your UID from `/etc/passwd` is propagated consistently during your session.

Your GID from `/etc/passwd` is considered your **primary GID**. This primary GID becomes both the real GID and the effective GID for your login shell. Your process also has a **group access list**, which includes your **secondary GIDs** from `/etc/group` plus your primary GID. When your process attempts *group* access to a file, the system checks your effective GID and and all the GIDs in your group access list looking for a match to the GID associated with the file. If any of your GIDs match that of the file, you are allowed *group* access to the file. Figure 4-2 illustrates this concept. Note that your primary GID is added to the group access list to facilitate file access in SGID programs, where your effective GID is no longer your primary GID.

Unless noted, any discussion of real and effective UIDs applies to real and effective GIDs.

Figure 4-2: GIDs and the Group Access List



To display your real UID and GID, use the `id` command. For example:

```
% id
uid=179(hale) gid=15(staff)
```

The `id` command displays the effective UID or GID only if different from the real UID or GID. In the following example, a process owned by `root` executes a program that changes the effective UID.

```
% id
uid=0(root) gid=1(daemon) euid=179(hale)
```

Because the effective UID determines *user* access rights, any process with your effective UID has your rights. If another user owns a process with your UID as the effective UID, your account is as much at risk as if that person knows your password. For this reason, do not execute an unknown program. Because you create the process that executes the program, the process would have your UID as its effective UID and can access your files and directories.

The following rules apply to any process that attempts to access a file:

- If the process's effective UID equals the file's UID, the process has the owner's access rights. The file's *user* permissions allow or deny access.
- If the process's effective GID, or any of the GIDs in the group access list, equals the file's GID, the process has the access rights of the group associated with the file. The file's *group* permissions allow or deny access.
- If the process's effective UID or GID does not equal the file's UID or GID, the file's *other* permissions allow or deny access.
- If the process's effective UID=0, the rules do not apply. Any process with an effective UID=0 has superuser privileges. It therefore has almost unlimited access to the operating system and user files. (For obvious reasons, a process whose effective UID=0 is called a **privileged process**.) The only limits are those placed by the system on operations that are considered unreasonable; for example, only a process that locks a file can unlock it. However, a process with an effective UID=0 could terminate the locking process and thereby release the lock.

The following section looks at how a process can have an effective UID or GID that differs from the real UID or GID.

4.3 SUID and SGID Programs

Set user ID (SUID) and set group ID (SGID) 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 grant a process the access rights of the file's owner. However, a poorly written or poorly protected SUID program presents a potential security risk, especially if the program is owned by `root`. Consider an SUID program owned by `root` that let a user escape to a shell. The user now has same privileges as the system manager, and the opportunity to exercise them.

The same logic applies to your account. Do not make a program that you own SUID or SGID without good reason. Under normal circumstances, ordinary users do not need SUID or SGID programs. If you must have an SUID or SGID program to provide controlled access to a file you own:

- Make sure that you know exactly what the program does. If you have any doubts, check with your security administrator before making the program SUID or SGID. Note that your security administrator can mount a file system `nosuid`, which allows an SUID or SGID program to execute, but does not allow the effective UID or GID to change from the real. This has the effect of reducing the program to a normal executable program.
- Make sure that only you can read or modify the program.
- Keep the program in a separate directory with restrictive permissions on the directory. Only you should have write permission to the directory. You may want to remove read permission for *group* and *other* to keep casual browsers from learning that you have SUID or SGID programs.

4.3.1 Example of an SUID Program

Whenever you change your password, you execute an SUID program. The `passwd` program is SUID to `root` to allow users to write to `/etc/passwd`, a file owned by `root`. Figure 4-3 shows the ownership and permissions on `/etc/passwd`; it also shows that an ordinary user does not have a UID or GID that would allow *user* or *group* access to `/etc/passwd`.

Figure 4-3: SUID Program

```
% ls -lg /etc/passwd
-rw-r--r--  1  root  system  4068  Apr 30  09:55   /etc/passwd

%cat /etc/passwd  |  grep root
root:KM8Fa5UT7NS3e:0:1:System PRIVILEGED Account,,,,:/:/bin/sh
                    ↑  ↑
          root's UID ┘  └ root's GID

%cat /etc/passwd  |  grep hale
hale:L8bE25dA4nm2:179:15:Harriet Hale:/usr/users/hale:/bin/csh
                      ↑   ↑
          hale's UID ┘    └ hale's GID
```

If there was no mechanism to allow users controlled access to certain system files like `/etc/passwd`, the following problem would exist:

- Only the owner of `/etc/passwd` could write to the password file. The file is owned by `root` (UID=0). Therefore any process that wants to modify this file must have an effective UID=0 to gain *user* access rights, or an effective GID=1 (system) to gain *group* access rights.
- The user in the example (hale) has UID=179 and GID=15. A process created by this user has an effective UID=179 and an effective GID=15. Suppose the user also belongs to three other groups from `/etc/group`, but none of these has GID=1. Because neither the UID nor and GIDs associated with the process

match those assigned to the /etc/passwd file, this user is considered *other* when file permissions are checked. Neither *group* nor *other* has write access to /etc/passwd.

- Therefore, the user cannot write to /etc/passwd. Because only the owner can write to the file, the security administrator (who has superuser privileges) would have to change passwords for all users.

However, because the passwd program is owned by root and is an SUID program, you can write to /etc/passwd and change your own password, but only under the control of the passwd program. Use the ls -l command to look at the permissions on the passwd program.

```
% ls -l /usr/bin/passwd

-rwsr-xr-x  1  root     system     17408  Oct 19  1988     /usr/bin/passwd
```

The letter "s" in the *user* execute permission indicates that the program is an SUID program. When you execute an SUID program, a child process is created with your real UID but with an effective UID of the owner of the SUID program. This child process now has the access rights of the owner of the SUID program (not a trivial issue when the SUID program is owned by root). When the child process exits, your parent process resumes execution and your effective UID is once again the same as your real UID.

When you execute passwd, your effective UID=0 because root owns /usr/bin/passwd. Because all file access to /etc/passwd takes place under program control, you cannot abuse superuser privileges while your child process is executing with its effective UID=0.

4.3.2 Example of an SGID Program

An SGID program is similar to an SUID program, but *group* access is in effect rather than *user* access. You must be a member of the group associated with an SGID file to run the program. A program can be both SUID and SGID; they are not mutually exclusive.

The following example shows the permissions on three programs that are both SUID and SGID.

```
% ls -lg /usr/ucb/lp*

-rws--s--x  1 root      daemon       50176 Oct 19  1988 /usr/ucb/lpq
-rws--s--x  1 root      daemon       44032 Oct 19  1988 /usr/ucb/lpr
-rws--s--x  1 root      daemon       53248 Oct 19  1988 /usr/ucb/lprm
```

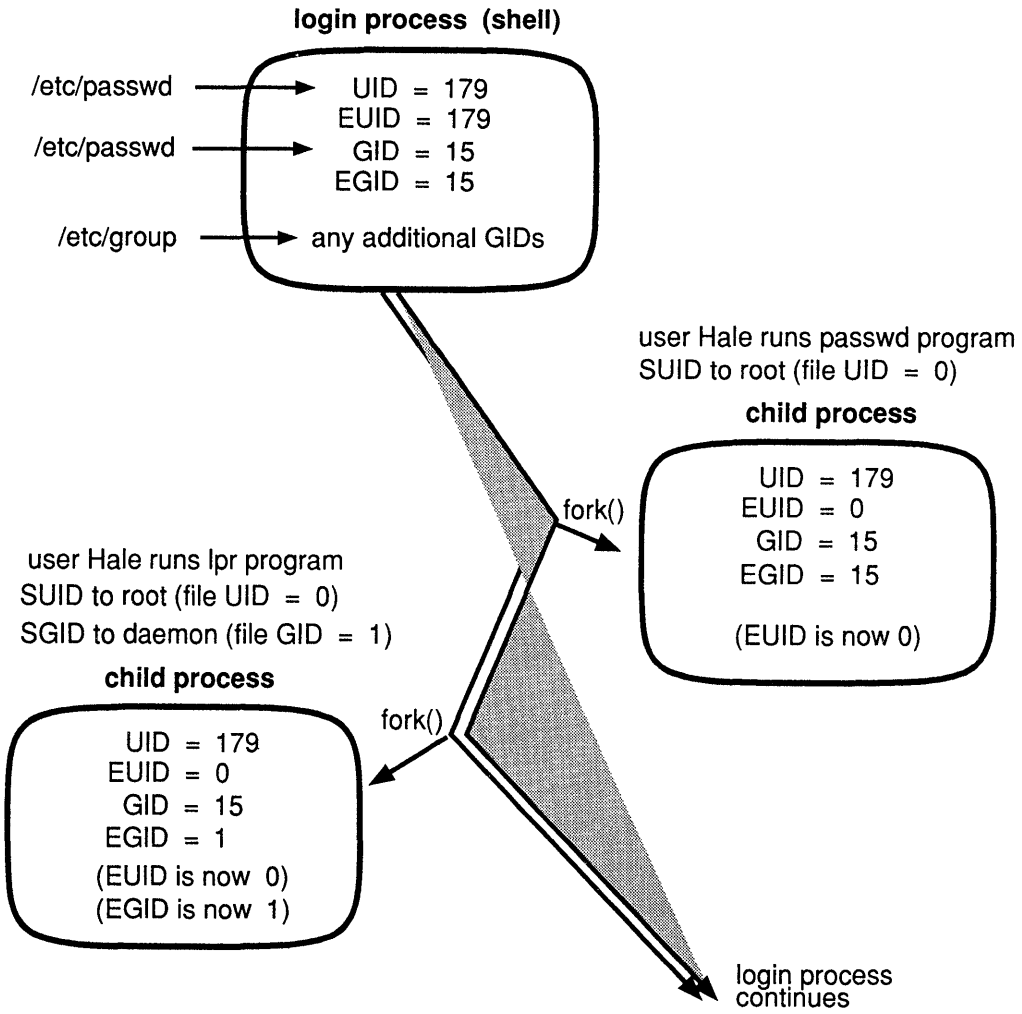
These are the programs you use to check print queues, send a file to a print queue, or remove a file from a print queue. They are SUID/SGID to allow you to put files in and remove files from directories that would normally deny you access. A process executing any of these commands has an effective UID=0 and an effective GID=1. These are the UID and GID assigned to the respective account entries for root and daemon in /etc/passwd.

```
% egrep '^root|^daemon' /etc/passwd

root:KM8Fa5UT7NS3e:0:1:System PRIVILEGED Account,,,,:/:/bin/csh
daemon:*:1:1:Mr Background:/:
```

Figure 4-4 shows how the `passwd` program changes a user's effective UID and how the `lpr` program changes a user's effective UID and effective GID.

Figure 4-4: SUID and SGID Programs



4.3.3 Copying and Moving SUID and SGID Programs

You can copy (`cp`) an SUID or SGID program from another user's account if you have read and execute permission on the source directory. The copy in your account retains its SUID/SGID permissions, but you become the owner. However, if the source file is owned by `root`, the system always removes SUID/SGID permission when the file is copied.

If you have a file with the same name as the SUID/SGID program, the copy takes the place of the original file, but the original permissions apply.

You can move (`mv`) an SUID or SGID program from another user's account to your account if you have write and execute permission on the source directory. If both files are on the same file system, the ownership is not changed, and SUID/SGID permissions are not removed. If the files are on different file systems, the ownership is changed, and the SUID/SGID permissions are removed.

4.4 Shells

Your login **shell** is the first process created with your UID and GID at login. In one sense, a shell is just another process; it has a PID, a PPID, and an owner. However, your shell is special because it is the environment in which you operate as a user.

4.4.1 Supported Shells

The ULTRIX operating system supports four shells:

- Bourne shell (sh)
- C shell (csh)
- sh5 shell (sh5)
- KornShell (ksh)

Each shell reads one or two startup files at login. Usually your security administrator installs a basic version of these startup files when creating your account. Most users modify these files to create a comfortable working environment. If you are not careful, you can unintentionally make your account vulnerable to the most simple forms of prying. However, with a little forethought and care these files not only create a comfortable environment, but a reasonably secure one.

Some shells read a second file when executed as a command. This allows the passing of environment variables that would not normally be inherited by a child process.

Bourne Shell	Reads `.profile` at login.
C Shell	Reads `.login` and `.cshrc` at login. Reads `.cshrc` when invoked after login.
sh5 Shell	Reads `.profile` at login.
KornShell	Reads `.profile` and `.kshrc` at login. Reads `.kshrc` when invoked after login.

4.4.2 Creating Secure Shell Startup Files

Your shell startup files are in your account and owned by you. The contents of these files and the order in which variables appear directly affect the security of your account.

Some basic rules for your shell startup files:

- Use absolute pathnames, ones that start with root (/). For example:

  ```
  PATH=/bin:/usr/bin/:/usr/local/bin:$HOME/bin:
  ```

  The shell searches the directories in the order listed in PATH. If you have a `$HOME/bin` directory, make sure that only you can write to it. Otherwise, someone could substitute a Trojan horse program for one of your programs. You would execute the Trojan horse program when you thought you were executing one of your own. (Trojan horse programs are described in Chapter 1.)

- Put the PATH variable before any other variables in your startup file to ensure absolute pathnames from the start of the session.

- Make sure that your PATH is passed to any processes you create. For example, in the Bourne shell:

  export PATH

  In the C shell, put PATH in your `.cshrc` file rather than your `.login` file. This ensures that the PATH variable is passed to processes started by your login shell.

- Do not put the current working directory (.) in PATH. Whenever you use the `cd` command to change to a directory, you could unknowingly execute a file placed in that directory to trap an unwary user. Putting the current working directory in your startup files makes every directory in the system a potential mine field.

- Do not put a directory like `/tmp` or `/usr/tmp` in PATH. Any user can put a file in these directories. A user could hide an executable file with the name of a real system command, hoping to trap someone whose PATH looks at `/tmp` before looking at the standard system directories where user commands are stored.

- Since these files are read by the login process, it is important to prevent them from being modified. Set the protections on any shell startup files to restrict read and write access to yourself. Unless there is a good reason for not doing so, remove both *group* and *other* access to these files. For example:

  ```
  % chmod 600 .profile
  ```

  To check the permission:

  ```
  % ls -l .profile
  -rw-------  1  hale          732  Mar 17   9:22    .profile
  ```

  Note that denying other users read permission does not mean that they cannot find your PATH or any other of your environment variables. The `-eaxww` options to the `ps` command display in wide format the environment variables for all processes on the system.

- Set a restrictive `umask` value (for example: **umask 027**) in your startup files.

The following example shows a template that you can use to begin creating a secure `.profile` file for the Bourne shell. Apply the same principles if you are creating a `.login` file for the C shell (remember to put PATH in `.cshrc` for the C shell).

```
#! /bin/sh
#
# use absolute pathnames in PATH
#
#(note: put your executables in $HOME/bin)
#
PATH=/bin:/usr/bin:/usr/local/bin:/usr/ucb:/etc:/usr/local/man:$HOME/bin
export PATH
#
# set a restrictive umask
#
umask 027
#
# don't allow other users to write to your terminal
#
mesg n
```

4.4.3 Shell Scripts

Many users create shell scripts to handle repetitive tasks. Since a shell script is a text file, it must provide read and execute access to whoever executes it. Because any user who can execute the script can also read it, a window of opportunity exists for sophisticated but unprincipled users. They could take advantage of the power inherent in the various shell programming languages to make the script perform actions that you did not intend. To reduce the risk of executing an illicit program, explicitly set the absolute search path for any program invoked through a shell script. Set restrictive permissions on any scripts you own.

Because a shell script is readable, it is inherently more open to compromise than a compiled program. If possible, create a compiled program rather than a shell script.

Do not make a shell script SUID. Because a shell script is less secure than a compiled program, you increase the risk that someone could compromise the script and gain control of a process that has your effective UID. Note that the security threat is even greater if you have superuser privileges and create an SUID shell script owned by `root`. A compromised SUID `root` shell script provides an attacker or malicious insider the potential to create a process with an effective UID=0.

4.5 Security Summary

Just as you own your files and directories, you own the processes you create. Because processes perform actions in your behalf (and with your privileges), you should know what rules govern a process's ability to read your files.

- Understand how the system identifies your processes, and the difference between real and effective UIDs and GIDs.
- Do not execute an unknown program. It will have your effective UID and GIDs — and therefore your rights.
- Use the `ps` command to keep track of your processes. This enables you to kill any suspicious processes, and alerts you to any runaway processes that are monopolizing system resources.
- Know how SUID and SGID programs work. Be aware of the potential risks when these mechanisms are used carelessly or when source files are poorly protected.
- Use your shell startup files to create a secure environment. Watch out for the more common dangers, such as using relative rather than absolute pathnames when defining the PATH variable. Put restrictive permissions on your shell startup files.
- Because shell scripts must be readable by those who execute them, they are less secure than compiled programs.

Connecting to Other Systems 5

While connecting systems gives users greater access to information, it also magnifies the security risks for each system. Responsible network security allows users some freedom while protecting valuable files from would-be attackers.

Your security administrator is responsible for most network security issues. However, individual users are responsible for being alert to security risks and protecting their accounts and files.

Four networking protocols enable ULTRIX users to communicate with users on remote systems:

- Internet protocols (TCP/IP, FTP, and TFTP)
- 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.

5.1 The rlogin, rcp, and rsh Commands

The TCP/IP protocol is the most commonly used networking protocol running under ULTRIX software. With TCP/IP, much of the network access to the computer is in the hands of users. TCP/IP commands that enable you to communicate with remote systems are:

`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 on the `rlogin` command, see `rlogin`(1c) in the *ULTRIX Reference Pages.*

`rcp` Lets you copy files to and from remote systems. For more information on the `rcp` command, see `rcp`(1c) in the *ULTRIX Reference Pages.*

`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 on the `rsh` command, see `rsh`(1c) in the *ULTRIX Reference Pages.*

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

5.1.1 The /etc/hosts.equiv File

The `/etc/hosts.equiv` file, which is owned by `root`, 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. For more information on the `hosts.equiv` file, see `hosts.equiv`(5yp) in the *ULTRIX Reference Pages.*

Because the `/etc/hosts.equiv` file is a remote system's key to your system, security-conscious security administrators leave this file empty or carefully restrict access to systems. If your security administrator leaves the `/etc/hosts.equiv` file empty, the only way that a user on a remote host can log in to the local host without typing a password is by logging in to your account, if his or her user name is listed in your `.rhosts` file.

5.1.2 The .rhosts File

The `.rhosts` file is a list of equivalent hosts that any user can create in his or her home directory. This file is the user counterpart of the `/etc/hosts.equiv` file. The `.rhosts` file has a narrower focus than its system-wide counterpart. The `/etc/hosts.equiv` file can affect the accounts of many users on a system, including yours. The `.rhosts` file affects only your own account.

Your `.rhosts` file enables users with your user name on equivalent hosts to log in to your account on the local host, without typing a password. The user must have a `.rhosts` file in his or her 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 a user name other than your user name next to the host name in your `.rhosts` file, that user can log in to your account on the local host. In this case, 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
```

The most common use of the `.rhosts` file is to simplify the 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.

Your .rhosts file can expand the access that the /etc/hosts.equiv file grants to your account. But, the .rhosts file 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 user from the /etc/hosts.equiv file, but you include that user in your .rhosts file, that user is considered trusted and can log in to your account without entering a password. The converse is not true. If your security administrator includes a user in the /etc/hosts.equiv file, you cannot exclude that user from accessing your account.

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.

  Chapter 3 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 security administrator.
- You should be the owner of your .rhosts file, and it must not be a symbolic link to another file.

5.2 The ftp Command

The ftp command enables you to transfer files to and from a remote site, using the ARPANET 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 on the `ftp` command, see `ftp`(1c) in the *ULTRIX Reference Pages.*

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 your system from a remote system with no audit trail. Security administrators concerned with network security avoid creating such anonymous accounts.

You should know and follow the security policy on using `ftp` for file transfers to remote systems. Talk to your security administrator about the security controls at your system. As mentioned previously in this chapter, use the `chmod` command to properly protect your files.

5.3 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. Since the `tftp` protocol does not validate user login information, setting proper permissions on your files is the only real protection from unauthorized access.

5.4 Local Area Transport (LAT) Commands

Your security administrator can increase the security of the LAT 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 (those connected to terminals, for example) while ignoring connections to all other LAT servers. For more information on using the LAT protocol, see `lcp`(8) in the *ULTRIX Reference Pages.*

Your security administrator can also set up LAT and hardwired terminals as secure terminals. If your security administrator has set up your terminal with a trusted path, you can press the Secure Attention Key (SAK) to log in to the host. Pressing this key kills any currently executing process prior to starting the login at your terminal. The login process proceeds as usual. The BREAK key may be configured as your Secure Attention Key. If your terminal connects to a terminal server that supports multiple sessions, and the BREAK key is normally used to display the terminal server prompt, you should remap this key. For example, to change the terminal server attention key on a DECserver 200 from BREAK to CTRL/A type the following at the `LOCAL>` prompt:

```
LOCAL> set port local CTRL/A
```

5.5 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 is widely adopted by most UNIX systems and enables you to transfer files between remote systems and the ULTRIX operating system. In addition, your system can use `uucp` to send and

receive mail across telephone lines.

Three uucp commands can present security concerns:

- uucp (UNIX-to-UNIX copy)
- tip
- cu

5.5.1 The uucp command

The uucp command is the UNIX-to-UNIX copy command. This 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 your security administrator has granted permission. For more information on the uucp command, see uucp(1c) in the *ULTRIX Reference Pages.* The uucp utility, left unrestricted, 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 security administrator exercises security measures when installing and setting up the uucp utility on your system. The following guidelines protect against unauthorized use of this powerful utility:

- 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 this 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. 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. For more information on setting the sticky bit, see the chmod(1) command in the *ULTRIX Reference Pages.* The following example sets the sticky bit on the documents directory:

```
% chmod 1777 documents
```

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

5.5.2 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. These commands connect your terminal to a modem on your system. You need only tell tip or cu what telephone number to call. For more information on the tip and cu commands, see tip(1c) and cu(1c) in the *ULTRIX Reference Pages.*

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.
- 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, check to see if your first session left any processes suspended, and kill those processes.

5.6 The dlogin, dls, and dcp Commands

If DECnet-ULTRIX 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 security 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:

```
dcp localfile remote_node/remote_user/remote_passwd::/remote_path/file
```

If you are connecting to a remote system that has no default user account, you must include this user name and 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. For more information on using these commands, see the *DECnet–ULTRIX User's and Programmer's Guide*.

5.7 Security Summary

Although security administrators are responsible for establishing and enforcing network security policy, network security is everyone's concern. You can protect your account and files by doing the following:

- Set appropriate permissions on your directories and files. For more information on setting file permissions, see Chapter 3.
- Be aware of the security policy for using your network and work closely with your security administrator to implement that policy in your account.

- Carefully control the contents of network access files, such as `.rhosts` and `.netrc`.
- Create a directory in your account for `uucp`, and use the `chmod` command to set the sticky bit on this directory.
- When you use the `tip` or `cu` command, never use an automatic login procedure and never leave your terminal while a remote session is in progress. 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.

Workstation and Windowing Environments 6

This chapter discusses DECwindows features that enhance the security of a workstation. This chapter does not explain how to use DECwindows. For information on using DECwindows, see the *DECwindows User's Guide.*

6.1 Who Can Access Your Workstation 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 could also 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. Imagine someone sending these commands to your workstation!

```
cd $HOME
rm -rf *
```

- **Copy.** Users on authorized hosts could also 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. A good rule of thumb is that if you can see it on your display, any user on an authorized host can see the same thing.

6.2 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, 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.

Thus, there are two ways to determine which hosts can access your workstation display:

- The system access control list
- The workstation access control list

6.2.1 The System Access Control List

Your security administrator can authorize a host to access a workstation's display by adding the host name to a system-wide 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 zero (0). 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.

The `/etc/X*.hosts` file is a system-wide access control list for your workstation. Each time you start a session on your workstation, the hosts that are named in this file are authorized to access your workstation display.

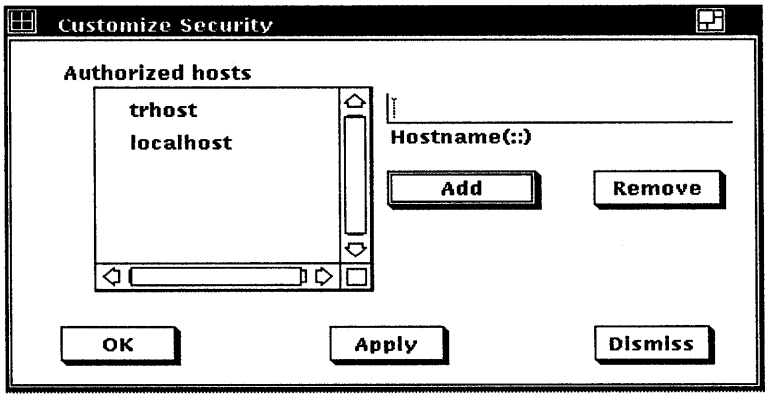
6.2.2 The Workstation Access Control List

Your workstation access control list can allow hosts access to your workstation display, even though the system access control list 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.

To authorize other users to use your workstation display:

1. Select the Session Manager window.
2. Select the Security... option from the Customize menu. The Customize Security box is displayed on the screen.



3. Type the host name you want to authorize.

4. Click on the Add button. The host name is added to the Authorized hosts box.
5. Click on either the OK or Apply button.

You can remove any host except the localhost (your workstation) from the access control list.

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 button.

Users logged in to the host you remove will no longer have access to your workstation for this session. However, the system access control list is checked each time you start a session. If there are conflicts between the system list and your workstation list, the system list prevails. This means that if you remove a host that your system list allows, the host will be returned to the workstation access control list the next time you start a session. Thus, removing a host is temporary if the host is listed in the `/etc/X*.hosts` file.

Storing the Workstation Access Control List

The changes you make to your workstation access control list 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 access control list.

Any user who can edit the `.Xdefaults` file could modify the access control list 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
```

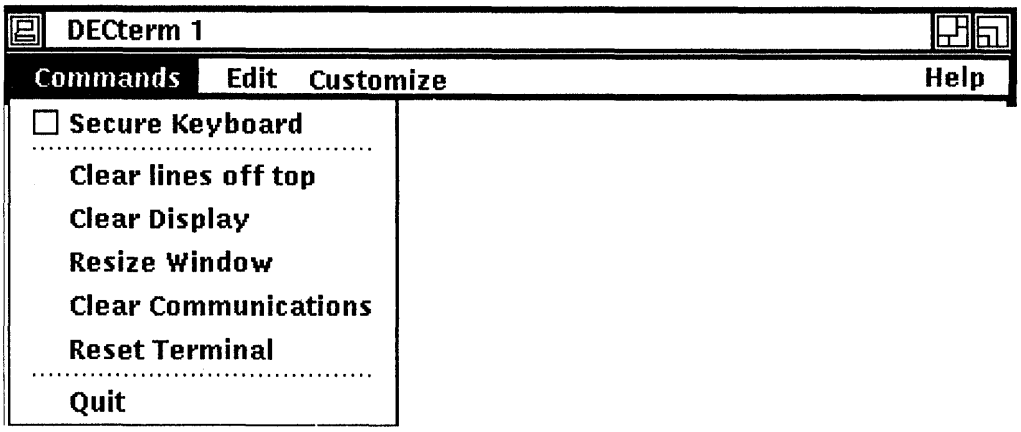
6.3 Protecting Keyboard Input

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.

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.

You can set secure keyboard mode by selecting the Secure Keyboard item from the Commands menu in a DECterm window.



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.

6.4 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
```

You should leave the `allowSendEvents` value set to **false**. This prevents unauthorized users from sending input into your DECterm window and executing commands under your user name.

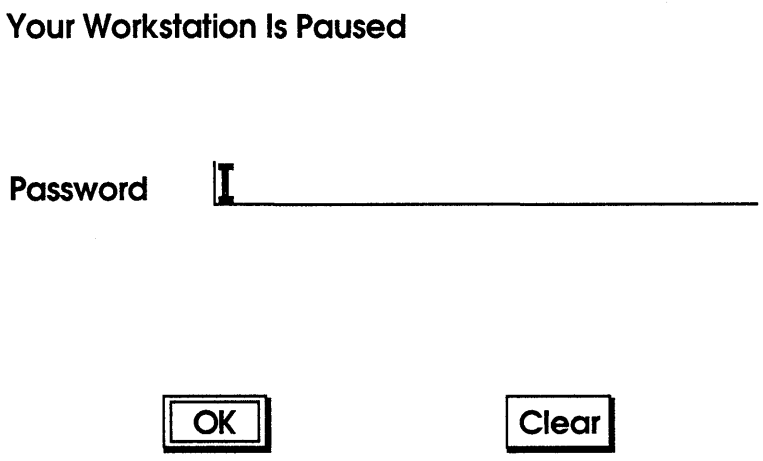
An application that opens its own window (not a DECterm window) might not block simulated keystrokes from your display. Therefore, if you are running such an application, check your access control list and remove any hosts that are authorized to access your display before working on sensitive files. 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.

6.5 Locking 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. Your can resume your session any time without having to recreate your screen environment.

Pausing a session in this way does not completely secure a workstation because anyone who boots the workstation into single-user mode can become the superuser. However, pausing a session is a reasonable precaution to take when you have to leave your workstation for a short period of time.

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:

1. Type your password.
2. Click on the OK button or press RETURN.

Once your password is verified, your session resumes.

6.6 Physical Security

Workstations often present problems for physical security; not because they are inherently less secure than other systems, but because workstations are typically found in ordinary offices, not in the more easily protected environment of the computer room.

In many cases, anyone who gains access to a workstation can easily get superuser status on that system. One method is to simply 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. Two steps can be taken to protect files on tape cartridges and floppy disks:

- Compress then encrypt files with the `compress` and `crypt` commands. Should an unauthorized person gain access to a disk, this step makes it difficult for the data to be read.
- Lock up all floppy disks and tape cartridges when they are not in use.

6.7 Security Summary

Workstations present some unique security considerations. Because they are typically located in offices, rather than computer rooms, it is difficult to protect them from unauthorized use.

Access control is the key to workstation security. A host that can access your workstation can execute commands as you. Any user logged in to an authorized host can access your workstation. Know which hosts are authorized to access your workstation display.

DECwindows includes some security features that you can use to improve the security of your workstation:

- Set the secure keyboard mode when using the `su` or `passwd` commands and typing your password, especially when another host is authorized to access your workstation display.
- Protect your `.Xdefaults` file. That file determines many window characteristics, including blocking keyboard and mouse input from other hosts. That file also lists the hosts that are authorized to access your workstation.
- Check your `.Xdefaults` file to make sure that the `allowSendEvents` resource is set to **false**.
- Be selective about the hosts you add to your access control list. If you are not sure whether to trust a host, check with your security administrator.
- Remember that a system-wide access control list may authorize a host to open a window on your workstation, even though you have not authorized the host access through the Customize Security menu.

Glossary A

absolute pathname

Pathnames beginning at `root` (/). *See also* **relative pathname**.

absolute file permissions

Permissions that replace existing permissions. Absolute permissions differ from **relative file permissions**, which add to or subtract from the current file permissions.

accountability

Identifying and holding a user accountable for actions that create, modify, provide access to, or disseminate information.

auditing

The chronological recording of events with security implications.

audit trail

The combination of all audited events.

authorization data base

The data base file `/etc/auth`, the optional ULTRIX `ndbm`(3) data base used to store passwords and password control information.

availability

Protecting the system from denial-of-service attacks that range from misusing system resources to crashing the system.

child process

A **process** created by another process using the `fork()` system call.

daemon

A long-lived, background **process** that performs a system-related service.

denial of service

The prevention of authorized access to system resources, or the delaying of time-critical operations.

directory file

A file that contains information about other files. The structure of a directory file is controlled by the system.

effective GID

The **GID** associated with the **process** that determines the group access rights of the process. The effective GID is usually the **real GID**, but can be changed through **SGID programs** and certain system calls and library routines.

effective UID

The **UID** associated with the **process** that determines the user access rights of the process. The effective UID is usually the **real UID**, but can be changed through SUID programs and certain system calls and library routines.

encryption

The process of transforming data to an unintelligible form in such a way that the original data either cannot be obtained (one-way encryption) or cannot be obtained without using the inverse decryption process (two-way encryption). The ULTRIX operating system uses one-way encryption for passwords.

file

A collection of related records treated as a unit and referenced by an **inode**.

file protection

The sum of all system processes and procedures designed to inhibit unauthorized access, modification, or destruction of a file.

file system

An initialized partition on a disk; a collection of files.

GID

Group ID. A unique integer assigned to a set of users.

group

A set of users assigned a unique group name and GID in the file `/etc/group`.

group access list

The list of group identifiers (**GID**s) associated with a **process**. The list includes the **primary GID** (from `/etc/passwd`) and any **secondary GID**s (from `/etc/group`).

hard link

A link to a file that is indistinguishable from the original directory entry (which is itself a hard link to the inode for the file). Any changes to the file are independent of the name used to reference the file. Hard links cannot exist between files on different file systems. *See also* **symbolic link**.

information security

Protecting the programs and information contained within the physical computing environment. *See also* **physical security**.

inode

A unique number that identifies a structure containing all the information about a file, except the file name and the actual contents of the file. Each file has an inode associated with it. Directory entries provide a cross-reference between file names and inodes.

integrity

The assurance that information is protected against unauthorized modification or destruction.

link

See **hard link, symbolic link**.

link count

The number of **hard links** to a file.

masquerade program

A program that pretends to be a legitimate portion of the operating environment in order to fool an authorized user into revealing sensitive information. For example, a program that mimics the login procedure in an attempt to steal passwords.

mode

See **permission**.

ordinary file

A file whose structure and content are controlled by users.

other

The file access category a **process** is placed in if it fails to be identified as either *user* or *group*.

parent process

A **process** that creates another process.

passphrase

A password composed of a string of words.

password

A character string that a user provides at login to validate identity as an authorized user of the system.

password aging

The removal of a password's effectiveness after a certain length of time.

password grabber

A type of **masquerade program** that mimics the login prompt and steals passwords.

permission

For file access, the authorization for *user*, *group*, or *other* to read, write, or execute the file. Also refers to setting the **SUID, SGID**, or **sticky bits** on a file.

physical security

Protecting a computer, its related peripherals, and media from physical attack. *See also* **information security**.

PID

Process ID. The unique identification number assigned to a **process** at its creation.

pipe

A mechanism that enables communication (in one direction) between **processes.**

PPID

Parent Process ID. The number associated with a **process** that identifies its **parent process.**

primary GID

The GID associated with a user account in `/etc/passwd` and a process's **group access list.** *See also* **GID, group access list, secondary GID.**

privileged process

A **process** whose **UID** is zero (UID=0).

process

The content (code) and context (environment) of an executing program.

real GID

The **GID** of the owner of the **process.**

real UID

The **UID** of the owner of the **process.**

relative pathname

A pathname that does not begin at `root` (/).

relative file permissions

Permissions that add to or subtract from the current file permissions. These differ from **absolute file permissions,** which replace current permissions.

root

Has three meanings: (1) The name of the user account with UID=0; (2) the top directory (/) in the file system; and (3) the file system containing the `root` directory.

secondary GID

One or more GIDs extracted from `/etc/group` and assigned to a process's **group access list.** *See also* **GID, group access list, primary GID.**

secrecy

Protecting information from unauthorized disclosure.

security administrator

The person responsible for implementing and maintaining system security.

SGID program

A program that changes the **effective GID** of the invoking **process** to the **GID** of the program.

shell

The program that acts as an interface between users and the operating system by interpreting and executing user commands.

shell escape

The mechanism that enables a user to issue a shell command from within a program.

shell script

An executable text file containing a series of shell commands.

socket

An endpoint for communication. For a description of the various socket types, see `socket`(2) in the *ULTRIX Reference Pages.*

soft expiration period

An option that allows a user one chance to log in with an expired password.

special file

A file that controls system input/output.

sticky bit

A file **permission**. When set on a sharable, executable file, the file will not be removed from the swap area. When set on a directory, only file owners, the directory owner, or the **superuser** can remove and rename files in the directory.

SUID program

A program that changes the **effective UID** of the invoking **process** to the **UID** of the program.

superuser

A user with access to the account whose UID=0. Typically, this account is named `root`. Any **process** with a UID=0 is considered a **privileged process** and is granted special privileges by the operating system.

symbolic link

Also called a soft link (as opposed to a **hard link**). Where a hard link points directly to the **inode** for a file, a symbolic link is a file that contains a pathname. Symbolic links can point to directories and to files on other file systems.

system manager

The person responsible for installing, configuring and maintaining the operating system and associated file systems. The manager may or may not act as **security administrator**.

Trojan horse program

A program that gains access to protected information under the pretext of performing some valid function. For example, a Trojan horse in the guise of a text editor could secretly copy user files to its creator's account.

trusted path

A mechanism by which a person at a terminal can communicate directly with a valid system program.

UID

User ID. A unique integer assigned to an individual user.

user

For file permissions, the owner of the file.

Security Summaries B

This appendix contains the security summaries from Chapters 2, 3, 4, 5, and 6.

B.1 Protecting Your Account

Your account is your first line of defense:

- If enabled on your system, use the Secure Attention Key to ensure that your user name and password are passed only to the correct system programs at login. This defeats attempts to steal this information.
- Pay attention to login messages that contradict your memory of your last login attempts. Use the `last` command to check recent logins to your account.
- Protect your password:
  - Pick a good one. Consider using a passphrase or a system-generated password.
  - Keep it secret. Do not write it down. Do not let others see you type it.
  - Change it regularly.
  - Use different passwords for different accounts.
- Lock your session whenever leaving your terminal during work hours.
- Verify that you have terminated all sessions before leaving work.

B.2 Protecting Your Files and Directories

Because you control permissions on your files, understand the range of available security options, and choose those options that best fit your security needs.

- Know how the system assigns owners and groups to files.
- Set a reasonably restrictive `umask` value in your shell startup files (for example, `umask 027`).
- Provide only the minimum file permissions required for your work. You only ask for trouble when everyone can access your files. Use the `chmod` command to modify permissions as needed.
- Use the group mechanism to control file access when you need to share files with other users.
- Create separate directories for files with similar security requirements.
- Use the `find` command to check for suspicious files and files with insecure permissions.

- Understand the interaction between certain system commands (such as `mv` or `cp`) and file permissions and ownership.
- Protect removable media. If someone copies your files into the system using the `cpio` command, that user becomes the owner of your files.
- Understand how links operate on files.
- Encrypt extremely sensitive information.

B.3 Processes and Shells

Just as you own your files and directories, you own the processes you create. Because processes perform actions in your behalf (and with your privileges), you should know what rules govern a process's ability to read your files.

- Understand how the system identifies your processes, and the difference between real and effective UIDs and GIDs.
- Do not execute an unknown program. It will have your effective UID and GIDs — and therefore your rights.
- Use the `ps` command to keep track of your processes. This enables you to kill any suspicious processes, and alerts you to any runaway processes that are monopolizing system resources.
- Know how SUID and SGID programs work. Be aware of the potential risks when these mechanisms are used carelessly or when source files are poorly protected.
- Use your shell startup files to create a secure environment. Watch out for the more common dangers, such as using relative rather than absolute pathnames when defining the PATH variable. Put restrictive permissions on your shell startup files.
- Because shell scripts must be readable by those who execute them, they are less secure than compiled programs.

B.4 Connecting to Other Systems

Although security administrators are responsible for establishing and enforcing network security policy, network security is everyone's concern. You can protect your account and files by doing the following:

- Set appropriate permissions on your directories and files. For more information on setting file permissions, see Chapter 3.
- Be aware of the security policy for using your network and work closely with your security administrator to implement that policy in your account.
- Carefully control the contents of network access files, such as `.rhosts` and `.netrc`.
- Create a directory in your account for `uucp`, and use the `chmod` command to set the sticky bit on this directory.
- When you use the `tip` or `cu` command, never use an automatic login procedure and never leave your terminal while a remote session is in progress. 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.

B.5 Workstation and Windowing Environments

Workstations present some unique security considerations. Because they are typically located in offices, rather than computer rooms, it is difficult to protect them from unauthorized use.

Access control is the key to workstation security. A host that can access your workstation can execute commands as you. Any user logged in to an authorized host can access your workstation. Know which hosts are authorized to access your workstation display.

DECwindows includes some security features that you can use to improve the security of your workstation:

- Set the secure keyboard mode when use the `su` or `passwd` commands and typing your password, especially when another host is authorized to access your workstation display.
- Protect your `.Xdefaults` file. That file determines many window characteristics, including blocking keyboard and mouse input from other hosts. That file also lists the hosts that are authorized to access your workstation.
- Check your `.Xdefaults` file to make sure that the **allowSendEvents** resource is set to false.
- Be selective about the hosts you add to your access control list. If you are not sure whether to trust a host, check with your security administrator.
- Remember that a system-wide access control list may authorize a host to open a window on your workstation, even though you have not authorized the host access through the Customize Security menu.

Index

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

W

X

Y

How to Order Additional Documentation

Technical Support

If you need help deciding which documentation best meets your needs, call 800-343-4040 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-baud 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*	————	SSB Order Processing - WMO/E15 *or* Software Supply Business Digital Equipment Corporation Westminster, Massachusetts 01473

* For internal orders, you must submit an Internal Software Order Form (EN-01740-07).

Reader's Comments

ULTRIX
Security Guide for Users
AA-PBKQA-TE

Please use this postage-paid form to comment on this manual. If you require a written reply to a software problem and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Thank you for your assistance.

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)	☐	☐	☐	☐
Page layout (easy to find information)	☐	☐	☐	☐

What would you like to see more/less of? _____

What do you like best about this manual? _____

What do you like least about this manual? _____

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 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
OPEN SOFTWARE PUBLICATIONS MANAGER
ZKO3–2/Z04
110 SPIT BROOK ROAD
NASHUA NH 03062–9987

Do Not Tear – Fold Here

Cut
Along
Dotted
Line

Reader's Comments

ULTRIX
Security Guide for Users
AA-PBKQA-TE

Please use this postage-paid form to comment on this manual. If you require a written reply to a software problem and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Thank you for your assistance.

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)	☐	☐	☐	☐
Page layout (easy to find information)	☐	☐	☐	☐

What would you like to see more/less of? _____

What do you like best about this manual? _____

What do you like least about this manual? _____

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 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
OPEN SOFTWARE PUBLICATIONS MANAGER
ZKO3–2/Z04
110 SPIT BROOK ROAD
NASHUA NH 03062–9987

Do Not Tear – Fold Here

Cut
Along
Dotted
Line