Final Report • April 1990

IMPROVING THE SECURITY OF YOUR UNIX SYSTEM

David A. Curry, Systems Programmer
Information and Telecommunications Sciences and Technology Division

ITSTD-721-FR-90-21

Approved:

Paul K. Hyder, Manager
Computer Facility

Boyd C. Fair, General Manager
Division Operations Section

Michael S. Frankel, Vice President
Information and Telecommunications Sciences and Technology Division

SRI International 333 Ravenswood Avenue • Menlo Park, CA 94025-3493 • (415) 326-6200 • FAX: (415) 326-5512 • Telex: 334486

SECTION 1
INTRODUCTION

1.1 UNIX SECURITY

The UNIX operating system, although now in widespread use in environments concerned about security, was not really designed with security in mind [Ritc75]. This does not mean that UNIX does not provide any security mechanisms; indeed, several very good ones are available. However, most ‘‘out of the box’’ installation procedures from companies such as Sun Microsystems still install the operating system in much the same way as it was installed 15 years ago: with little or no security enabled.

The reasons for this state of affairs are largely historical. UNIX was originally designed by programmers for use by other programmers. The environment in which it was used was one of open cooperation, not one of privacy. Programmers typically collaborated with each other on projects, and hence preferred to be able to share their files with each other without having to climb over security hurdles. Because the first sites outside of Bell Laboratories to install UNIX were university research laboratories, where a similar environment existed, no real need for greater security was seen until some time later.

In the early 1980s, many universities began to move their UNIX systems out of the research laboratories and into the computer centers, allowing (or forcing) the user population as a whole to use this new and wonderful system. Many businesses and government sites began to install UNIX systems as well, particularly as desktop workstations became more powerful and affordable. Thus, the UNIX operating system is no longer being used only in environments where open collaboration is the goal. Universities require their students to use the system for class assignments, yet they do not want the students to be able to copy from each other. Businesses use their UNIX systems for confidential tasks such as bookkeeping and payroll. And the government uses UNIX systems for various unclassified yet sensitive purposes.

To complicate matters, new features have been added to UNIX over the years, making security even more difficult to control. Perhaps the most problematic features are those relating to networking: remote login, remote command execution, network file systems, diskless workstations, and electronic mail. All of these features have increased the utility and usability of UNIX by untold amounts. However, these same features, along with the widespread connection of UNIX systems to the Internet and other networks, have opened up many new areas of vulnerability to unauthorized abuse of the system.

UNIX is a registered trademark of AT&T. VAX is a trademark of Digital Equipment Corporation. Sun-3 and NFS are trademarks of Sun Microsystems. Annex is a trademark of Xylogics, Inc.

1.2 THE INTERNET WORM

On the evening of November 2, 1988, a self-replicating program, called a *worm*, was released on the Internet [Seel88, Spaf88, Eich89]. Overnight, this program had copied itself from machine to machine, causing the machines it infected to labor under huge loads, and denying service to the users of those machines. Although the program only infected two types of computers,* it spread quickly, as did the concern, confusion, and sometimes panic of system administrators whose machines were affected. While many system administrators were aware that something like this could theoretically happen – the security holes exploited by the worm were well known – the scope of the worm's break-ins came as a great surprise to most people.

The worm itself did not destroy any files, steal any information (other than account passwords), intercept private mail, or plant other destructive software [Seel88]. However, it did manage to severely disrupt the operation of the network. Several sites, including parts of MIT, NASA's Ames Research Center and Goddard Space Flight Center, the Jet Propulsion Laboratory, and the U. S. Army Ballistic Research Laboratory, disconnected themselves from the Internet to avoid recontamination. In addition, the Defense Communications Agency ordered the connections between the MILNET and ARPANET shut down, and kept them down for nearly 24 hours [Eich89, Elme88]. Ironically, this was perhaps the worst thing to do, since the first fixes to combat the worm were distributed via the network [Eich89].

This incident was perhaps the most widely described computer security problem ever. The worm was covered in many newspapers and magazines around the country including the *New York Times*, *Wall Street Journal*, *Time* and most computer-oriented technical publications, as well as on all three major television networks, the Cable News Network, and National Public Radio. In January 1990, a United States District Court jury found Robert Tappan Morris, the author of the worm, guilty of charges brought against him under a 1986 federal computer fraud and abuse law. Morris faces up to five years in prison and a $250,000 fine [Schu90]. Sentencing is scheduled for May 4, 1990.

1.3 SPIES AND ESPIONAGE

In August 1986, the Lawrence Berkeley Laboratory, an unclassified research laboratory at the University of California at Berkeley, was attacked by an unauthorized computer intruder [Stol88, Stol89]. Instead of immediately closing the holes the intruder was using, the system administrator, Clifford Stoll, elected to watch the intruder and document the weaknesses he exploited. Over the next 10 months, Stoll watched the intruder attack over 400 computers around the world, and successfully enter about 30. The computers broken into were located at universities, military bases, and defense contractors [Stol88].

---

* Sun-3 systems from Sun Microsystems and VAX systems from Digital Equipment Corp., both running variants of 4.*x* BSD UNIX from the University of California at Berkeley.

Unlike many intruders seen on the Internet, who typically enter systems and browse around to see what they can, this intruder was looking for something specific. Files and data dealing with the Strategic Defense Initiative, the space shuttle, and other military topics all seemed to be of special interest. Although it is unlikely that the intruder would have found any truly classified information (the Internet is an unclassified network), it was highly probable that he could find a wealth of sensitive material [Stol88].

After a year of tracking the intruder (eventually involving the FBI, CIA, National Security Agency, Air Force Intelligence, and authorities in West Germany), five men in Hannover, West Germany were arrested. In March 1989, the five were charged with espionage: they had been selling the material they found during their exploits to the KGB. One of the men, Karl Koch (''Hagbard''), was later found burned to death in an isolated forest outside Hannover. No suicide note was found [Stol89]. In February 1990, three of the intruders (Markus Hess, Dirk Bresinsky, and Peter Carl) were convicted of espionage in a German court and sentenced to prison terms, fines, and the loss of their rights to participate in elections [Risk90]. The last of the intruders, Hans Hübner (''Pengo''), still faces trial in Berlin.

1.4 OTHER BREAK-INS

Numerous other computer security problems have occurred in recent years, with varying levels of publicity. Some of the more widely known incidents include break-ins on NASA's SPAN network [McLe87], the IBM ''Christmas Virus'' [Risk87], a virus at Mitre Corp. that caused the MILNET to be temporarily isolated from other networks [Risk88], a worm that penetrated DECNET networks [Risk89a], break-ins on U. S. banking networks [Risk89b], and a multitude of viruses, worms, and trojan horses affecting personal computer users.

1.5 SECURITY IS IMPORTANT

As the previous stories demonstrate, computer security is an important topic. This document describes the security features provided by the UNIX operating system, and how they should be used. The discussion centers around version 4.*x* of SunOS, the version of UNIX sold by Sun Microsystems. Most of the information presented applies equally well to other UNIX systems. Although there is no way to make a computer completely secure against unauthorized use (other than to lock it in a room and turn it off), by following the instructions in this document you can make your system impregnable to the ''casual'' system cracker,* and make it more difficult for the sophisticated cracker to penetrate.

---

* The term ''hacker,'' as applied to computer users, originally had an honorable connotation: ''a person who enjoys learning the details of programming systems and how to stretch their capabilities - as opposed to most users of computers, who prefer to learn only the minimum amount necessary'' [Stee88]. Unfortunately, the media has distorted this definition and given it a dishonorable meaning. In deference to the true hackers, we will use the term ''cracker'' throughout this document.

SECTION 2
IMPROVING SECURITY

UNIX system security can be divided into three main areas of concern. Two of these areas, account security and network security, are primarily concerned with keeping unauthorized users from gaining access to the system. The third area, file system security, is concerned with preventing unauthorized access, either by legitimate users or crackers, to the data stored in the system. This section describes the UNIX security tools provided to make each of these areas as secure as possible.

2.1 ACCOUNT SECURITY

One of the easiest ways for a cracker to get into a system is by breaking into someone's account. This is usually easy to do, since many systems have old accounts whose users have left the organization, accounts with easy-to-guess passwords, and so on. This section describes methods that can be used to avoid these problems.

2.1.1 Passwords

The password is the most vital part of UNIX account security. If a cracker can discover a user's password, he can then log in to the system and operate with all the capabilities of that user. If the password obtained is that of the super-user, the problem is more serious: the cracker will have read and write access to every file on the system. For this reason, choosing secure passwords is extremely important.

The UNIX *passwd* program [Sun88a, 379] places very few restrictions on what may be used as a password. Generally, it requires that passwords contain five or more lowercase letters, or four characters if a nonalphabetic or uppercase letter is included. However, if the user ''insists'' that a shorter password be used (by entering it three times), the program will allow it. No checks for obviously insecure passwords (see below) are performed. Thus, it is incumbent upon the system administrator to ensure that the passwords in use on the system are secure.

In [Morr78], the authors describe experiments conducted to determine typical users' habits in the choice of passwords. In a collection of 3,289 passwords, 16% of them contained three characters or less, and an astonishing 86% were what could generally be described as insecure. Additional experiments in [Gram84] show that by trying three simple guesses on each account – the login name, the login name in reverse, and the two concatenated together – a cracker can expect to obtain access to between 8 and 30 percent of the accounts on a typical system. A second experiment showed that by trying the 20 most common female first names, followed by a single digit (a total of 200 passwords), at least one password was valid on each of several dozen machines surveyed. Further experimentation by the author has found that by trying variations on the login name, user's first and last names, and a list of nearly 1800 common first names, up to 50 percent of the passwords on any given system can be cracked in a matter of two or three days.

2.1.1.1 Selecting Passwords

The object when choosing a password is to make it as difficult as possible for a cracker to make educated guesses about what you've chosen. This leaves him no alternative but a brute-force search, trying every possible combination of letters, numbers, and punctuation. A search of this sort, even conducted on a machine that could try one million passwords per second (most machines can try less than one hundred per second), would require, on the average, over one hundred years to complete. With this as our goal, and by using the information in the preceding text, a set of guidelines for password selection can be constructed:

- **Don't** use your login name in any form (as-is, reversed, capitalized, doubled, etc.).
- **Don't** use your first or last name in any form.
- **Don't** use your spouse's or child's name.
- **Don't** use other information easily obtained about you. This includes license plate numbers, telephone numbers, social security numbers, the brand of your automobile, the name of the street you live on, etc.
- **Don't** use a password of all digits, or all the same letter. This significantly decreases the search time for a cracker.
- **Don't** use a word contained in (English or foreign language) dictionaries, spelling lists, or other lists of words.
- **Don't** use a password shorter than six characters.
- **Do** use a password with mixed-case alphabetics.
- **Do** use a password with nonalphabetic characters, e.g., digits or punctuation.
- **Do** use a password that is easy to remember, so you don't have to write it down.
- **Do** use a password that you can type quickly, without having to look at the keyboard. This makes it harder for someone to steal your password by watching over your shoulder.

Although this list may seem to restrict passwords to an extreme, there are several methods for choosing secure, easy-to-remember passwords that obey the above rules. Some of these include the following:

- Choose a line or two from a song or poem, and use the first letter of each word. For example, ‘‘In Xanadu did Kubla Kahn a stately pleasure dome decree’’ becomes ‘‘IXdKKaspdd.’’
- Alternate between one consonant and one or two vowels, up to eight characters. This provides nonsense words that are usually pronounceable, and thus easily remembered. Examples include ‘‘routboo,’’ ‘‘quadpop,’’ and so on.
- Choose two short words and concatenate them together with a punctation character between them. For example: ‘‘dog;rain,’’ ‘‘book+mug,’’ ‘‘kid?goat.’’

The importance of obeying these password selection rules cannot be overemphasized. The Internet worm, as part of its strategy for breaking into new machines, attempted to crack user passwords. First, the worm tried simple choices such as the login name, user's first and last names, and so on. Next, the worm tried each word present in an internal dictionary of 432 words (presumably Morris considered these words to be ‘‘good’’ words to try). If all else failed, the worm tried going through the system dictionary, */usr/dict/words*, trying each word [Spaf88]. The password selection rules above successfully guard against all three of these strategies.

2.1.1.2 Password Policies

Although asking users to select secure passwords will help improve security, by itself it is not enough. It is also important to form a set of password policies that all users must obey, in order to keep the passwords secure.

First and foremost, it is important to impress on users the need to keep their passwords in their minds only. Passwords should never be written down on desk blotters, calendars, and the like. Further, storing passwords in files on the computer must be prohibited. In either case, by writing the password down on a piece of paper or storing it in a file, the security of the user's account is totally dependent on the security of the paper or file, which is usually less than the security offered by the password encryption software.

A second important policy is that users must never give out their passwords to others. Many times, a user feels that it is easier to give someone else his password in order to copy a file, rather than to set up the permissions on the file so that it can be copied. Unfortunately, by giving out the password to another person, the user is placing his trust in this other person not to distribute the password further, write it down, and so on.

Finally, it is important to establish a policy that users must change their passwords from time to time, say twice a year. This is difficult to enforce on UNIX, since in most implementations, a password-expiration scheme is not available. However, there are ways to implement this policy, either by using third-party software or by sending a memo to the users requesting that they change their passwords.

This set of policies should be printed and distributed to all current users of the system. It should also be given to all new users when they receive their accounts. The policy usually carries more weight if you can get it signed by the most ‘‘impressive’’ person in your organization (e.g., the president of the company).

2.1.1.3 Checking Password Security

The procedures and policies described in the previous sections, when properly implemented, will greatly reduce the chances of a cracker breaking into your system via a stolen account. However, as with all security measures, you as the system administrator must periodically check to be sure that the policies and procedures are being adhered to. One of the unfortunate truisms of password security is that, ‘‘left to their own ways, some people will still use cute doggie names as passwords’’ [Gram84].

The best way to check the security of the passwords on your system is to use a password-cracking program much like a real cracker would use. If you succeed in cracking any passwords, those passwords should be changed immediately. There are a few freely available password cracking programs distributed via various source archive sites; these are described in more detail in Section 4. A fairly extensive cracking program is also available from the author. Alternatively, you can write your own cracking program, and tailor it to your own site. For a list of things to check for, see the list of guidelines above.

2.1.2 Expiration Dates

Many sites, particularly those with a large number of users, typically have several old accounts lying around whose owners have since left the organization. These accounts are a major security hole: not only can they be broken into if the password is insecure, but because nobody is using the account anymore, it is unlikely that a break-in will be noticed.

The simplest way to prevent unused accounts from accumulating is to place an expiration date on every account. These expiration dates should be near enough in the future that old accounts will be deleted in a timely manner, yet far enough apart that the users will not become annoyed. A good figure is usually one year from the date the account was installed. This tends to spread the expirations out over the year, rather than clustering them all at the beginning or end. The expiration date can easily be stored in the password file (in the full name field). A simple shell script can be used to periodically check that all accounts have expiration dates, and that none of the dates has passed.

On the first day of each month, any user whose account has expired should be contacted to be sure he is still employed by the organization, and that he is actively using the account. Any user who cannot be contacted, or who has not used his account recently, should be deleted from the system. If a user is unavailable for some reason (e.g., on vacation) and cannot be contacted, his account should be disabled by replacing the encrypted password in the password file entry with an asterisk (*). This makes it impossible to log in to the account, yet leaves the account available to be re-enabled on the user’s return.

2.1.3 Guest Accounts

Guest accounts present still another security hole. By their nature, these accounts are rarely used, and are always used by people who should only have access to the machine for the short period of time they are guests. The most secure way to handle guest accounts is to install them on an as-needed basis, and delete them as soon as the people using them leave. Guest accounts should never be given simple passwords such as ‘‘guest’’ or ‘‘visitor,’’ and should never be allowed to remain in the password file when they are not being used.

2.1.4 Accounts Without Passwords

Some sites have installed accounts with names such as ‘‘who,’’ ‘‘date,’’ ‘‘lpq,’’ and so on that execute simple commands. These accounts are intended to allow users to execute these commands without having to log in to the machine. Typically these accounts have no password associated with them, and can thus be used by anyone. Many of the accounts are given a user id of zero, so that they execute with super-user permissions.

The problem with these accounts is that they open potential security holes. By not having passwords on them, and by having super-user permissions, these accounts practically invite crackers to try to penetrate them. Usually, if the cracker can gain access to the system, penetrating these accounts is simple, because each account executes a different command. If the cracker can replace any one of these commands with one of his own, he can then use the unprotected account to execute his program with super-user permissions.

Simply put, accounts without passwords should not be allowed on any UNIX system.

2.1.5 Group Accounts and Groups

Group accounts have become popular at many sites, but are actually a break-in waiting to happen. A group account is a single account shared by several people, e.g., by all the collaborators on a project. As mentioned in the section on password security, users should not share passwords – the group account concept directly violates this policy. The proper way to allow users to share information, rather than giving them a group account to use, is to place these users into a group. This is done by editing the group file, */etc/group* [Sun88a, 1390; Sun88b, 66], and creating a new group with the users who wish to collaborate listed as members.

A line in the group file looks like

```
groupname:password:groupid:user1,user2,user3,...
```

The *groupname* is the name assigned to the group, much like a login name. It may be the same as someone’s login name, or different. The maximum length of a group name is eight characters. The password field is unused in BSD-derived versions of UNIX, and should contain an asterisk (*). The *groupid* is a number from 0 to 65535 inclusive. Generally, numbers below 10 are reserved for special purposes, but you may choose any unused number. The last field is a comma-separated (no spaces) list of the login names of the users in the group. If no login names are listed, then the group has no members. To create a group called ‘‘hackers’’ with Huey, Duey, and Louie as members, you would add a line such as this to the group file:

```
hackers:*:100:huey,duey,louie
```

After the group has been created, the files and directories the members wish to share can then be changed so that they are owned by this group, and the group permission bits on the files and directories can be set to allow sharing. Each user retains his own account, with his own password, thus protecting the security of the system.

For example, to change Huey's ‘‘programs’’ directory to be owned by the new group and properly set up the permissions so that all members of the group may access it, the *chgrp* and *chmod* commands would be used as follows [Sun88a, 63-66]:

```
# chgrp hackers ~huey/programs
# chmod -R g+rw ~huey/programs
```

2.1.6 Yellow Pages

The Sun Yellow Pages system [Sun88b, 349-374] allows many hosts to share password files, group files, and other files via the network, while the files are stored on only a single host. Unfortunately, Yellow Pages also contains a few potential security holes.

The principal way Yellow Pages works is to have a special line in the password or group file that begins with a ‘‘+’’. In the password file, this line looks like

```
+::0:0:::
```

and in the group file, it looks like

```
+:
```

These lines should only be present in the files stored on Yellow Pages client machines. They should not be present in the files on the Yellow Pages master machine(s). When a program reads the password or group file and encounters one of these lines, it goes through the network and requests the information it wants from the Yellow Pages server instead of trying to find it in the local file. In this way, the data does not have to be maintained on every host. Since the master machine already has all the information, there is no need for this special line to be present there.

Generally speaking, the Yellow Pages service itself is reasonably secure. There are a few openings that a sophisticated (and dedicated) cracker could exploit, but Sun is rapidly closing these. The biggest problem with Yellow Pages is the ‘‘+’’ line in the password file. If the ‘‘+’’ is deleted from the front of the line, then this line loses its special Yellow Pages meaning. It instead becomes a regular password file line for an account with a null login name, no password, and user id zero (super-user). Thus, if a careless system administrator accidentally deletes the ‘‘+’’. the whole system is wide open to any attack.*

Yellow Pages is too useful a service to suggest turning it off, although turning it off would make your system more secure. Instead, it is recommended that you read carefully the information in the Sun manuals in order to be fully aware of Yellow Pages' abilities and its limitations.

* Actually, a line like this without a ‘‘+’’ is dangerous in any password file, regardless of whether Yellow Pages is in use.

2.2 NETWORK SECURITY

As trends toward internetworking continue, most sites will, if they haven't already, connect themselves to one of the numerous regional networks springing up around the country. Most of these regional networks are also interconnected, forming the Internet [Hind83, Quar86]. This means that the users of your machine can access other hosts and communicate with other users around the world. Unfortunately, it also means that other hosts and users from around the world can access your machine, and attempt to break into it.

Before internetworking became commonplace, protecting a system from unauthorized access simply meant locking the machine in a room by itself. Now that machines are connected by networks, however, security is much more complex. This section describes the tools and methods available to make your UNIX networks as secure as possible.

2.2.1 Trusted Hosts

One of the most convenient features of the Berkeley (and Sun) UNIX networking software is the concept of ''trusted'' hosts. The software allows the specification of other hosts (and possibly users) who are to be considered trusted – remote logins and remote command executions from these hosts will be permitted without requiring the user to enter a password. This is very convenient, because users do not have to type their password every time they use the network. Unfortunately, for the same reason, the concept of a trusted host is also extremely insecure.

The Internet worm made extensive use of the trusted host concept to spread itself throughout the network [Seel88]. Many sites that had already disallowed trusted hosts did fairly well against the worm compared with those sites that did allow trusted hosts. Even though it is a security hole, there are some valid uses for the trusted host concept. This section describes how to properly implement the trusted hosts facility while preserving as much security as possible.

2.2.1.1 The hosts.equiv File

The file */etc/hosts.equiv* [Sun88a, 1397] can be used by the system administrator to indicate trusted hosts. Each trusted host is listed in the file, one host per line. If a user attempts to log in (using *rlogin*) or execute a command (using *rsh*) remotely from one of the systems listed in *hosts.equiv*, and that user has an account on the local system with the same login name, access is permitted without requiring a password.

Provided adequate care is taken to allow only local hosts in the *hosts.equiv* file, a reasonable compromise between security and convenience can be achieved. Nonlocal hosts (including hosts at remote sites of the same organization) should never be trusted. Also, if there are any machines at your organization that are installed in ''public'' areas (e.g., terminal rooms) as opposed to private offices, you should not trust these hosts.

On Sun systems, *hosts.equiv* is controlled with the Yellow Pages software. As distributed, the default *hosts.equiv* file distributed by Sun contains a single line:

```
+
```

This indicates that *every known host* (i.e., the complete contents of the host file) should be considered a trusted host. This is totally incorrect and a major security hole, since hosts outside the local organization should never be trusted. A correctly configured *hosts.equiv* should never list any ‘‘wildcard’’ hosts (such as the ‘‘+’’); only specific host names should be used. When installing a new system from Sun distribution tapes, you should be sure to either replace the Sun default *hosts.equiv* with a correctly configured one, or delete the file altogether.

2.2.1.2 The .rhosts File

The *.rhosts* file [Sun88a, 1397] is similar in concept and format to the *hosts.equiv* file, but allows trusted access only to specific host-user combinations, rather than to hosts in general.* Each user may create a *.rhosts* file in his home directory, and allow access to her account without a password. Most people use this mechanism to allow trusted access between accounts they have on systems owned by different organizations who do not trust each other's hosts in *hosts.equiv*. Unfortunately, this file presents a major security problem: While *hosts.equiv* is under the system administrator's control and can be managed effectively, any user may create a *.rhosts* file granting access to whomever he chooses, without the system administrator's knowledge.

The only secure way to manage *.rhosts* files is to completely disallow them on the system. The system administrator should check the system often for violations of this policy (see Section 3.3.1.4). One possible exception to this rule is the ‘‘root’’ account; a *.rhosts* file may be necessary to allow network backups and the like to be completed.

2.2.2 Secure Terminals

Under newer versions of UNIX, the concept of a ‘‘secure’’ terminal has been introduced. Simply put, the super-user (‘‘root’’) may not log in on a nonsecure terminal, even with a password. (Authorized users may still use the *su* command to become super-user, however.) The file */etc/ttytab* [Sun88a, 1478] is used to control which terminals are considered secure.† A short excerpt from this file is shown below.

---

* Actually, *hosts.equiv* may be used to specify host-user combinations as well, but this is rarely done.

† Under non-Sun versions of Berkeley UNIX, this file is called */etc/ttys*.

```
console  "/usr/etc/getty std.9600"  sun      off secure
ttya     "/usr/etc/getty std.9600"  unknown  off secure
ttyb     "/usr/etc/getty std.9600"  unknown  off secure
ttyp0    none                       network  off secure
ttyp1    none                       network  off secure
ttyp2    none                       network  off secure
```

The keyword ‘‘secure’’ at the end of each line indicates that the terminal is considered secure. To remove this designation, simply edit the file and delete the ‘‘secure’’ keyword. After saving the file, type the command (as super-user):

```
# kill -HUP 1
```

This tells the *init* process to reread the *ttytab* file.

The Sun default configuration for *ttytab* is to consider all terminals secure, including ‘‘pseudo’’ terminals used by the remote login software. This means that ‘‘root’’ may log in remotely from any host on the network. A more secure configuration would consider as secure only directly connected terminals, or perhaps only the console device. This is how file servers and other machines with disks should be set up.

The most secure configuration is to remove the ‘‘secure’’ designation from all terminals, including the console device. This requires that those users with super-user authority first log in as themselves, and then become the super-user via the *su* command. It also requires the ‘‘root’’ password to be entered when rebooting in single-user mode, in order to prevent users from rebooting their desktop workstations and obtaining super-user access. This is how all diskless client machines should be set up.

2.2.3 The Network File System

The Network File System (NFS) [Sun88d] is designed to allow several hosts to share files over the network. One of the most common uses of NFS is to allow diskless workstations to be installed in offices, while keeping all disk storage in a central location. As distributed by Sun, NFS has no security features enabled. This means that any host on the Internet may access your files via NFS, regardless of whether you trust them or not.

Fortunately, there are several easy ways to make NFS more secure. The more commonly used methods are described in this section, and these can be used to make your files quite secure from unauthorized access via NFS. Secure NFS, introduced in SunOS Release 4.0, takes security one step further, using public-key encryption techniques to ensure authorized access. Discussion of secure NFS is deferred until Section 4.

2.2.3.1 The exports File

The file */etc/exports* [Sun88a, 1377] is perhaps one of the most important parts of NFS configuration. This file lists which file systems are exported (made available for mounting) to other systems. A typical *exports* file as installed by the Sun installation procedure looks something like this:

```
/usr
/home
/var/spool/mail
#
/export/root/client1    -access=client1,root=client1
/export/swap/client1    -access=client1,root=client1
#
/export/root/client2    -access=client2,root=client2
/export/swap/client2    -access=client2,root=client2
```

The *root=* keyword specifies the list of hosts that are allowed to have super-user access to the files in the named file system. This keyword is discussed in detail in Section 2.2.3.3. The *access=* keyword specifies the list of hosts (separated by colons) that are allowed to mount the named file system. If no *access=* keyword is specified for a file system, any host anywhere on the network may mount that file system via NFS.

Obviously, this presents a major security problem, since anyone who can mount your file systems via NFS can then peruse them at her leisure. Thus, it is important that all file systems listed in *exports* have an *access=* keyword associated with them. If you have only a few hosts which must mount a file system, you can list them individually in the file:

```
/usr    -access=host1:host2:host3:host4:host5
```

However, because the maximum number of hosts that can be listed this way is ten, the *access=* keyword will also allow netgroups to be specified. Netgroups are described in the next section.

After making any changes to the *exports* file, you should run the command

```
# exportfs -a
```

in order to make the changes take effect.

2.2.3.2 The netgroup File

The file */etc/netgroup* [Sun88a, 1407] is used to define netgroups. This file is controlled by Yellow Pages, and must be rebuilt in the Yellow Pages maps whenever it is modified. Consider the following sample *netgroup* file:

```
A_Group      (servera,,) (clienta1,,) (clienta2,,)

B_Group      (serverb,,) (clientb1,,) (clientb2,,)

AdminStaff   (clienta1,mary,) (clientb3,joan,)

AllSuns      A_Group B_Group
```

This file defines four netgroups, called *A_Group*, *B_Group*, *AdminStaff*, and *AllSuns*. The *AllSuns* netgroup is actually a ‘‘super group’’ containing all the members of the *A_Group* and *B_Group* netgroups.

Each member of a netgroup is defined as a triple: (host, user, domain). Typically, the *domain* field is never used, and is simply left blank. If either the *host* or *user* field is left blank, then any host or user is considered to match. Thus the triple (host,,) matches any user on the named host, while the triple (,user,) matches the named user on any host.

Netgroups are useful when restricting access to NFS file systems via the *exports* file. For example, consider this modified version of the file from the previous section:

```
/usr                    -access=A_Group
/home                   -access=A_Group:B_Group
/var/spool/mail         -access=AllSuns
#
/export/root/client1    -access=client1,root=client1
/export/swap/client1    -access=client1,root=client1
#
/export/root/client2    -access=client2,root=client2
/export/swap/client2    -access=client2,root=client2
```

The */usr* file system may now only be mounted by the hosts in the *A_Group* netgroup, that is, *servera*, *clienta1*, and *clienta2*. Any other host that tries to mount this file system will receive an ‘‘access denied’’ error. The */home* file system may be mounted by any of the hosts in either the *A_Group* or *B_Group* netgroups. The */var/spool/mail* file system is also restricted to these hosts, but in this example we used the ‘‘super group’’ called *AllSuns*.

Generally, the best way to configure the *netgroup* file is to make a single netgroup for each file server and its clients, and then to make other super groups, such as *AllSuns*. This allows you the flexibility to specify the smallest possible group of hosts for each file system in */etc/exports*.

Netgroups can also be used in the password file to allow access to a given host to be restricted to the members of that group, and they can be used in the *hosts.equiv* file to centralize maintenance of the list of trusted hosts. The procedures for doing this are defined in more detail in the Sun manual.

2.2.3.3 Restricting Super-User Access

Normally, NFS translates the super-user id to a special id called ‘‘nobody’’ in order to prevent a user with ‘‘root’’ on a remote workstation from accessing other people’s files. This is good for security, but sometimes a nuisance for system administration, since you cannot make changes to files as ‘‘root’’ through NFS.

The *exports* file also allows you to grant super-user access to certain file systems for certain hosts by using the *root=* keyword. Following this keyword a colon-separated list of up to ten hosts may be specified; these hosts will be allowed to access the file system as ‘‘root’’ without having the user id converted to ‘‘nobody.’’ Netgroups may not be specified to the *root=* keyword.

Granting ‘‘root’’ access to a host should not be done lightly. If a host has ‘‘root’’ access to a file system, then the super-user on that host will have complete access to the file system, just as if you had given him the ‘‘root’’ password on the server. Untrusted hosts should never be given ‘‘root’’ access to NFS file systems.

2.2.4 FTP

The File Transfer Protocol, implemented by the *ftp* and *ftpd* programs [Sun88a, 195-201, 1632-1634], allows users to connect to remote systems and transfer files back and forth. Unfortunately, older versions of these programs also had several bugs in them that allowed crackers to break into a system. These bugs have been fixed by Berkeley, and new versions are available. If your *ftpd* * was obtained before December 1988, you should get a newer version (see Section 4).

One of the more useful features of FTP is the ‘‘anonymous’’ login. This special login allows users who do not have an account on your machine to have restricted access in order to transfer files from a specific directory. This is useful if you wish to distribute software to the public at large without giving each person who wants the software an account on your machine. In order to securely set up anonymous FTP you should follow the specific instructions below:

1. Create an account called ‘‘ftp.’’ Disable the account by placing an asterisk (*) in the password field. Give the account a special home directory, such as */usr/ftp* or */usr/spool/ftp*.
2. Make the home directory owned by ‘‘ftp’’ and unwritable by anyone:

```
# chown ftp ~ftp
# chmod 555 ~ftp
```

3. Make the directory *~ftp/bin*, owned by the super-user and unwritable by anyone. Place a copy of the *ls* program in this directory:

---

* On Sun systems, *ftpd* is stored in the file */usr/etc/in.ftpd*. On most other systems, it is called */etc/ftpd*.

```
# mkdir ~ftp/bin
# chown root ~ftp/bin
# chmod 555 ~ftp/bin
# cp -p /bin/ls ~ftp/bin
# chmod 111 ~ftp/bin/ls
```

4. Make the directory *~ftp/etc*, owned by the super-user and unwritable by anyone. Place copies of the password and group files in this directory, with all the password fields changed to asterisks (*). You may wish to delete all but a few of the accounts and groups from these files; the only account that must be present is ‘‘ftp.’’

```
# mkdir ~ftp/etc
# chown root ~ftp/etc
# chmod 555 ~ftp/etc
# cp -p /etc/passwd /etc/group ~ftp/etc
# chmod 444 ~ftp/etc/passwd ~ftp/etc/group
```

5. Make the directory *~ftp/pub*, owned by ‘‘ftp’’ and world-writable. Users may then place files that are to be accessible via anonymous FTP in this directory:

```
# mkdir ~ftp/pub
# chown ftp ~ftp/pub
# chmod 777 ~ftp/pub
```

Because the anonymous FTP feature allows anyone to access your system (albeit in a very limited way), it should not be made available on every host on the network. Instead, you should choose one machine (preferably a server or standalone host) on which to allow this service. This makes monitoring for security violations much easier. If you allow people to transfer files to your machine (using the world-writable *pub* directory, described above), you should check often the contents of the directories into which they are allowed to write. Any suspicious files you find should be deleted.

2.2.4.1 Trivial FTP

The Trivial File Transfer Protocol, TFTP, is used on Sun workstations (and others) to allow diskless hosts to boot from the network. Basically, TFTP is a stripped-down version of FTP – there is no user authentication, and the connection is based on the User Datagram Protocol instead of the Transmission Control Protocol. Because they are so stripped-down, many implementations of TFTP have security holes. You should check your hosts by executing the command sequence shown below.

```
% tftp
tftp> connect yourhost
tftp> get /etc/motd tmp
Error code 1: File not found
tftp> quit
%
```

If your version does not respond with ‘‘*File not found*,’’ and instead transfers the file, you should replace your version of *tftpd* * with a newer one. In particular, versions of SunOS prior to release 4.0 are known to have this problem.

2.2.5 Mail

Electronic mail is one of the main reasons for connecting to outside networks. On most versions of Berkeley-derived UNIX systems, including those from Sun, the *sendmail* program [Sun88a, 1758-1760; Sun88b, 441-488] is used to enable the receipt and delivery of mail. As with the FTP software, older versions of *sendmail* have several bugs that allow security violations. One of these bugs was used with great success by the Internet worm [Seel88, Spaf88]. The current version of *sendmail* from Berkeley is version 5.61, of January 1989. Sun is, as of this writing, still shipping version 5.59, which has a known security problem. They have, however, made a fixed version available. Section 4 details how to obtain these newer versions.

Generally, with the exception of the security holes mentioned above, *sendmail* is reasonably secure when installed by most vendors' installation procedures. There are, however, a few precautions that should be taken to ensure secure operation:

1. Remove the ‘‘decode’’ alias from the aliases file (*/etc/aliases* or */usr/lib/aliases*).
2. If you create aliases that allow messages to be sent to programs, be absolutely sure that there is no way to obtain a shell or send commands to a shell from these programs.
3. Make sure the ‘‘wizard’’ password is disabled in the configuration file, *sendmail.cf*. (Unless you modify the distributed configuration files, this shouldn't be a problem.)
4. Make sure your *sendmail* does not support the ‘‘debug’’ command. This can be done with the following commands:

* On Sun systems, *tftpd* is stored in the file */usr/etc/in.tftpd*. On most other systems, it is called */etc/tftpd*.

```
% telnet localhost 25
220 yourhost Sendmail 5.61 ready at 9 Mar 90 10:57:36 PST
debug
500 Command unrecognized
quit
%
```

If your *sendmail* responds to the ‘‘debug’’ command with ‘‘*200 Debug set*,’’ then you are vulnerable to attack and should replace your *sendmail* with a newer version.

By following the procedures above, you can be sure that your mail system is secure.

2.2.6 Finger

The ‘‘finger’’ service, provided by the *finger* program [Sun88a, 186-187], allows you to obtain information about a user such as her full name, home directory, last login time, and in some cases when she last received mail and/or read her mail. The *fingerd* program [Sun88a, 1625] allows users on remote hosts to obtain this information.

A bug in *fingerd* was also exercised with success by the Internet worm [Seel88, Spaf88]. If your version of *fingerd** is older than November 5, 1988, it should be replaced with a newer version. New versions are available from several of the sources described in Section 4.

2.2.7 Modems and Terminal Servers

Modems and terminal servers (terminal switches, Annex boxes, etc.) present still another potential security problem. The main problem with these devices is one of configuration – misconfigured hardware can allow security breaches. Explaining how to configure every brand of modem and terminal server would require volumes. However, the following items should be checked for on any modems or terminal servers installed at your site:

1. If a user dialed up to a modem hangs up the phone, the system should log him out. If it doesn’t, check the hardware connections and the kernel configuration of the serial ports.
2. If a user logs off, the system should force the modem to hang up. Again, check the hardware connections if this doesn’t work.
3. If the connection from a terminal server to the system is broken, the system should log the user off.

* On Sun systems, *fingerd* is stored in */usr/etc/in.fingerd*. On most other systems, it is called */etc/fingerd*.

4. If the terminal server is connected to modems, and the user hangs up, the terminal server should inform the system that the user has hung up.

Most modem and terminal server manuals cover in detail how to properly connect these devices to your system. In particular you should pay close attention to the ''Carrier Detect,'' ''Clear to Send,'' and ''Request to Send'' connections.

2.2.8 Firewalls

One of the newer ideas in network security is that of a *firewall*. Basically, a firewall is a special host that sits between your outside-world network connection(s) and your internal network(s). This host does not send out routing information about your internal network, and thus the internal network is ''invisible'' from the outside. In order to configure a firewall machine, the following considerations need to be taken:

1. The firewall does not advertise routes. This means that users on the internal network must log in to the firewall in order to access hosts on remote networks. Likewise, in order to log in to a host on the internal network from the outside, a user must first log in to the firewall machine. This is inconvenient, but more secure.
2. All electronic mail sent by your users must be forwarded to the firewall machine if it is to be delivered outside your internal network. The firewall must receive all incoming electronic mail, and then redistribute it. This can be done either with aliases for each user or by using name server MX records.
3. The firewall machine should not mount any file systems via NFS, or make any of its file systems available to be mounted.
4. Password security on the firewall must be rigidly enforced.
5. The firewall host should not trust any other hosts regardless of where they are. Furthermore, the firewall should not be trusted by any other host.
6. Anonymous FTP and other similar services should only be provided by the firewall host, if they are provided at all.

The purpose of the firewall is to prevent crackers from accessing other hosts on your network. This means, in general, that you must maintain strict and rigidly enforced security on the firewall, but the other hosts are less vulnerable, and hence security may be somewhat lax. But it is important to remember that the firewall is not a complete cure against crackers – if a cracker can break into the firewall machine, he can then try to break into any other host on your network.

2.3 FILE SYSTEM SECURITY

The last defense against system crackers are the permissions offered by the file system. Each file or directory has three sets of permission bits associated with it: one set for the user who owns the file, one set for the users in the group with which the file is associated, and one set for all other users (the ‘‘world’’ permissions). Each set contains three identical permission bits, which control the following:

read If set, the file or directory may be read. In the case of a directory, read access allows a user to see the contents of a directory (the names of the files contained therein), but not to access them.

write If set, the file or directory may be written (modified). In the case of a directory, write permission implies the ability to create, delete, and rename files. Note that the ability to remove a file is *not* controlled by the permissions on the file, but rather the permissions on the directory containing the file.

execute If set, the file or directory may be executed (searched). In the case of a directory, execute permission implies the ability to access files contained in that directory.

In addition, a fourth permission bit is available in each set of permissions. This bit has a different meaning in each set of permission bits:

setuid If set in the owner permissions, this bit controls the ‘‘set user id’’ (setuid) status of a file. Setuid status means that when a program is executed, it executes with the permissions of the user owning the program, in addition to the permissions of the user executing the program. For example, *sendmail* is setuid ‘‘root,’’ allowing it to write files in the mail queue area, which normal users are not allowed to do. This bit is meaningless on nonexecutable files.

setgid If set in the group permissions, this bit controls the ‘‘set group id’’ (setgid) status of a file. This behaves in exactly the same way as the setuid bit, except that the group id is affected instead. This bit is meaningless on nonexecutable files (but see below).

sticky If set in the world permissions, the ‘‘sticky’’ bit tells the operating system to do special things with the text image of an executable file. It is mostly a holdover from older versions of UNIX, and has little if any use today. This bit is also meaningless on nonexecutable files (but see below).

2.3.1 Setuid Shell Scripts

Shell scripts that have the setuid or setgid bits set on them are *not* secure, regardless of how many safeguards are taken when writing them. There are numerous software packages available that claim to make shell scripts secure, but every one released so far has not managed to solve all the problems.

Setuid and setgid shell scripts should never be allowed on any UNIX system.

2.3.2 The Sticky Bit on Directories

Newer versions of UNIX have attached a new meaning to the sticky bit. When this bit is set on a directory, it means that users may not delete or rename other users' files in this directory. This is typically useful for the */tmp* directory. Normally, */tmp* is world-writable, enabling any user to delete another user's files. By setting the sticky bit on */tmp*, users may only delete their own files from this directory.

To set the sticky bit on a directory, use the command

chmod o+t *directory*

2.3.3 The Setgid Bit on Directories

In SunOS 4.0, the setgid bit was also given a new meaning. Two rules can be used for assigning group ownership to a file in SunOS:

1. The System V mechanism, which says that a user's primary group id (the one listed in the password file) is assigned to any file he creates.
2. The Berkeley mechanism, which says that the group id of a file is set to the group id of the directory in which it is created.

If the setgid bit is set on a directory, the Berkeley mechanism is enabled. Otherwise, the System V mechanism is enabled. Normally, the Berkeley mechanism is used; this mechanism must be used if creating directories for use by more than one member of a group (see Section 2.1.5).

To set the setgid bit on a directory, use the command

chmod g+s *directory*

2.3.4 The umask Value

When a file is created by a program, say a text editor or a compiler, it is typically created with all permissions enabled. Since this is rarely desirable (you don't want other users to be able to write your files), the *umask* value is used to modify the set of permissions a file is created with. Simply put, while the *chmod* command [Sun88a, 65-66] specifies what bits should be turned *on*, the umask value specifies what bits should be turned *off*.

For example, the default umask on most systems is 022. This means that write permission for the group and world should be turned off whenever a file is created. If instead you wanted to turn off all group and world permission bits, such that any file you created would not be readable, writable, or executable by anyone except yourself, you would set your umask to 077.

The umask value is specified in the *.cshrc* or *.profile* files read by the shell using the *umask* command [Sun88a, 108, 459]. The ''root'' account should have the line

```
umask 022
```

in its */.cshrc* file, in order to prevent the accidental creation of world-writable files owned by the super-user.

2.3.5 Encrypting Files

The standard UNIX *crypt* command [Sun88a, 95] is not at all secure. Although it is reasonable to expect that *crypt* will keep the casual ‘‘browser’’ from reading a file, it will present nothing more than a minor inconvenience to a determined cracker. *Crypt* implements a one-rotor machine along the lines of the German Enigma (broken in World War II). The methods of attack on such a machine are well known, and a sufficiently large file can usually be decrypted in a few hours even without knowledge of what the file contains [Reed84]. In fact, publicly available packages of programs designed to ‘‘break’’ files encrypted with *crypt* have been around for several years.

There are software implementations of another algorithm, the Data Encryption Standard (DES), available on some systems. Although this algorithm is much more secure than *crypt*, it has never been proven that it is totally secure, and many doubts about its security have been raised in recent years.

Perhaps the best thing to say about encrypting files on a computer system is this: if you think you have a file whose contents are important enough to encrypt, then that file should not be stored on the computer in the first place. This is especially true of systems with limited security, such as UNIX systems and personal computers.

It is important to note that UNIX passwords are *not* encrypted with the *crypt* program. Instead, they are encrypted with a modified version of the DES that generates one-way encryptions (that is, the password cannot be decrypted). When you log in, the system does not decrypt your password. Instead, it encrypts your attempted password, and if this comes out to the same result as encrypting your real password, you are allowed to log in.

2.3.6 Devices

The security of devices is an important issue in UNIX. Device files (usually residing in */dev*) are used by various programs to access the data on the disk drives or in memory. If these device files are not properly protected, your system is wide open to a cracker. The entire list of devices is too long to go into here, since it varies widely from system to system. However, the following guidelines apply to all systems:

1. The files */dev/kmem*, */dev/mem*, and */dev/drum* should never be readable by the world. If your system supports the notion of the ‘‘kmem’’ group (most newer systems do) and utilities such as *ps* are setgid ‘‘kmem,’’ then these files should be owned by user ‘‘root’’ and group ‘‘kmem,’’ and should be mode 640. If your system does not support the notion of the ‘‘kmem’’ group, and utilities such as *ps* are setuid ‘‘root,’’ then these files should be owned by user ‘‘root’’ and mode 600.

2. The disk devices, such as */dev/sd0a*, */dev/rxy1b*, etc., should be owned by user ‘‘root’’ and group ‘‘operator,’’ and should be mode 640. Note that each disk has eight partitions and two device files for each partition. Thus, the disk ‘‘sd0’’ would have the following device files associated with it in */dev*:

sd0a	sd0e	rsd0a	rsd0e
sd0b	sd0f	rsd0b	rsd0f
sd0c	sd0g	rsd0c	rsd0g
sd0d	sd0h	rsd0d	rsd0h

3. With very few exceptions, all other devices should be owned by user ‘‘root.’’ One exception is terminals, which are changed to be owned by the user currently logged in on them. When the user logs out, the ownership of the terminal is automatically changed back to ‘‘root.’’

2.4 SECURITY IS YOUR RESPONSIBILITY

This section has detailed numerous tools for improving security provided by the UNIX operating system. The most important thing to note about these tools is that although they are available, they are typically not put to use in most installations. Therefore, it is incumbent on you, the system administrator, to take the time and make the effort to enable these tools, and thus to protect your system from unauthorized access.

SECTION 3
MONITORING SECURITY

One of the most important tasks in keeping any computer system secure is monitoring the security of the system. This involves examining system log files for unauthorized accesses of the system, as well as monitoring the system itself for security holes. This section describes the procedures for doing this. An additional part of monitoring security involves keeping abreast of security problems found by others; this is described in Section 5.

3.1 ACCOUNT SECURITY

Account security should be monitored periodically in order to check for two things: users logged in when they ‘‘shouldn’t’’ be (e.g., late at night, when they’re on vacation, etc.), and users executing commands they wouldn’t normally be expected to use. The commands described in this section can be used to obtain this information from the system.

3.1.1 The lastlog File

The file */usr/adm/lastlog* [Sun88a, 1485] records the most recent login time for each user of the system. The message printed each time you log in, e.g.,

```
Last login: Sat Mar 10 10:50:48 from spam.itstd.sri.c
```

uses the time stored in the *lastlog* file. Additionally, the last login time reported by the *finger* command uses this time. Users should be told to carefully examine this time whenever they log in, and to report unusual login times to the system administrator. This is an easy way to detect account break-ins, since each user should remember the last time she logged into the system.

3.1.2 The utmp and wtmp Files

The file */etc/utmp* [Sun88a, 1485] is used to record who is currently logged into the system. This file can be displayed using the *who* command [Sun88a, 597]:

```
% who
hendra   tty0c   Mar 13 12:31
heidari  tty14   Mar 13 13:54
welgem   tty36   Mar 13 12:15
reagin   ttyp0   Mar 13 08:54   (aaifs.itstd.sri.)
ghg      ttyp1   Mar  9 07:03   (hydra.riacs.edu)
compion  ttyp2   Mar  1 03:01   (ei.ecn.purdue.ed)
```

For each user, the login name, terminal being used, login time, and remote host (if the user is logged in via the network) are displayed.

The file */usr/adm/wtmp* [Sun88a, 1485] records each login and logout time for every user. This file can also be displayed using the *who* command:

```
% who /usr/adm/wtmp
davy     ttyp4    Jan  7 12:42 (annex01.riacs.ed)
         ttyp4    Jan  7 15:33
davy     ttyp4    Jan  7 15:33 (annex01.riacs.ed)
         ttyp4    Jan  7 15:35
hyder    ttyp3    Jan  8 09:07 (triceratops.itst)
         ttyp3    Jan  8 11:43
```

A line that contains a login name indicates the time the user logged in; a line with no login name indicates the time that the terminal was logged off. Unfortunately, the output from this command is rarely as simple as in the example above; if several users log in at once, the login and logout times are all mixed together and must be matched up by hand using the terminal name.

The *wtmp* file may also be examined using the *last* command [Sun88a, 248]. This command sorts out the entries in the file, matching up login and logout times. With no arguments, *last* displays all information in the file. By giving the name of a user or terminal, the output can be restricted to the information about the user or terminal in question. Sample output from the *last* command is shown below.

```
% last
davy      ttyp3  intrepid.itstd.s Tue Mar 13 10:55 - 10:56 (00:00)
hyder     ttyp3  clyde.itstd.sri. Mon Mar 12 15:31 - 15:36 (00:04)
reboot    ~                       Mon Mar 12 15:16
shutdown  ~                       Mon Mar 12 15:16
arms      ttyp3  clyde0.itstd.sri Mon Mar 12 15:08 - 15:12 (00:04)
hyder     ttyp3  spam.itstd.sri.c Sun Mar 11 21:08 - 21:13 (00:04)
reboot    ~                       Sat Mar 10 20:05
davy      ftp    hydra.riacs.edu  Sat Mar 10 13:23 - 13:30 (00:07)
```

For each login session, the user name, terminal used, remote host (if the user logged in via the network), login and logout times, and session duration are shown. Additionally, the times of all system shutdowns and reboots (generated by the *shutdown* and *reboot* commands [Sun88a, 1727, 1765]) are recorded. Unfortunately, system crashes are not recorded. In newer versions of the operating system, pseudo logins such as those via the *ftp* command are also recorded; an example of this is shown in the last line of the sample output, above.

3.1.3 The acct File

The file */usr/adm/acct* [Sun88a, 1344-1345] records each execution of a command on the system, who executed it, when, and how long it took. This information is logged each time a command completes, but only if your kernel was compiled with the SYSACCT option enabled (the option is enabled in some GENERIC kernels, but is usually disabled by default).

The *acct* file can be displayed using the *lastcomm* command [Sun88a, 249]. With no arguments, all the information in the file is displayed. However, by giving a command name, user name, or terminal name as an argument, the output can be restricted to information about the given command, user, or terminal. Sample output from *lastcomm* is shown below.

```
% lastcomm
sh          S     root      __        0.67 secs Tue Mar 13 12:45
atrun             root      __        0.23 secs Tue Mar 13 12:45
lpd          F    root      __        1.06 secs Tue Mar 13 12:44
lpr         S     burwell  tty09      1.23 secs Tue Mar 13 12:44
troff             burwell  tty09     12.83 secs Tue Mar 13 12:44
eqn               burwell  tty09      1.44 secs Tue Mar 13 12:44
df                kindred  ttyq7      0.78 secs Tue Mar 13 12:44
ls                kindred  ttyq7      0.28 secs Tue Mar 13 12:44
cat               kindred  ttyq7      0.05 secs Tue Mar 13 12:44
stty              kindred  ttyq7      0.05 secs Tue Mar 13 12:44
tbl               burwell  tty09      1.08 secs Tue Mar 13 12:44
rlogin      S     jones    ttyp3      5.66 secs Tue Mar 13 12:38
rlogin       F    jones    ttyp3      2.53 secs Tue Mar 13 12:41
stty              kindred  ttyq7      0.05 secs Tue Mar 13 12:44
```

The first column indicates the name of the command. The next column displays certain flags on the command: an ‘‘F’’ means the process spawned a child process, ‘‘S’’ means the process ran with the set-user-id bit set, ‘‘D’’ means the process exited with a core dump, and ‘‘X’’ means the process was killed abnormally. The remaining columns show the name of the user who ran the program, the terminal he ran it from (if applicable), the amount of CPU time used by the command (in seconds), and the date and time the process started.

3.2 NETWORK SECURITY

Monitoring network security is more difficult, because there are so many ways for a cracker to attempt to break in. However, there are some programs available to aid you in this task. These are described in this section.

3.2.1 The syslog Facility

The *syslog* facility [Sun88a, 1773] is a mechanism that enables any command to log error messages and informational messages to the system console, as well as to a log file. Typically, error messages are logged in the file */usr/adm/messages* along with the date, time, name of the program sending the message, and (usually) the process id of the program. A sample segment of the *messages* file is shown below.

```
Mar 12 14:53:37 sparkyfs login: ROOT LOGIN ttyp3 FROM setekfs.itstd.sr
Mar 12 15:18:08 sparkyfs login: ROOT LOGIN ttyp3 FROM setekfs.itstd.sr
Mar 12 16:50:25 sparkyfs login: ROOT LOGIN ttyp4 FROM pongfs.itstd.sri
Mar 12 16:52:20 sparkyfs vmunix: sd2c:  read failed, no retries
Mar 13 06:01:18 sparkyfs vmunix: /: file system full
Mar 13 08:02:03 sparkyfs login: ROOT LOGIN ttyp4 FROM triceratops.itst
Mar 13 08:28:52 sparkyfs su: davy on /dev/ttyp3
Mar 13 08:38:03 sparkyfs login: ROOT LOGIN ttyp4 FROM triceratops.itst
Mar 13 10:56:54 sparkyfs automount[154]: host aaifs not responding
Mar 13 11:30:42 sparkyfs login: REPEATED LOGIN FAILURES ON ttyp3 FROM
                intrepid.itstd.s, daemon
```

Of particular interest in this sample are the messages from the *login* and *su* programs. Whenever someone logs in as ‘‘root,’’ *login* logs this information. Generally, logging in as ‘‘root’’ directly, rather than using the *su* command, should be discouraged, as it is hard to track which person is actually using the account. Once this ability has been disabled, as described in Section 2.2.2, detecting a security violation becomes a simple matter of searching the *messages* file for lines of this type.

Login also logs any case of someone repeatedly trying to log in to an account and failing. After three attempts, *login* will refuse to let the person try anymore. Searching for these messages in the *messages* file can alert you to a cracker attempting to guess someone’s password.

Finally, when someone uses the *su* command, either to become ‘‘root’’ or someone else, *su* logs the success or failure of this operation. These messages can be used to check for users sharing their passwords, as well as for a cracker who has penetrated one account and is trying to penetrate others.

3.2.2 The showmount Command

The *showmount* command [Sun88a, 1764] can be used on an NFS file server to display the names of all hosts that currently have something mounted from the server. With no options, the program simply displays a list of all the hosts. With the *-a* and *-d* options, the output is somewhat more useful. The first option, *-a*, causes *showmount* to list all the host and directory combinations. For example,

```
bronto.itstd.sri.com:/usr/share
bronto.itstd.sri.com:/usr/local.new
bronto.itstd.sri.com:/usr/share/lib
bronto.itstd.sri.com:/var/spool/mail
cascades.itstd.sri.com:/sparky/a
clyde.itstd.sri.com:/laser_dumps
cm1.itstd.sri.com:/sparky/a
coco0.itstd.sri.com:/sparky/a
```

There will be one line of output for each directory mounted by a host. With the *-d* option, *showmount* displays a list of all directories that are presently mounted by some host.

The output from *showmount* should be checked for two things. First, only machines local to your organization should appear there. If you have set up proper netgroups as described in Section 2.2.3, this should not be a problem. Second, only ''normal'' directories should be mounted. If you find unusual directories being mounted, you should find out who is mounting them and why – although it is probably innocent, it may indicate someone trying to get around your security mechanisms.

3.3 FILE SYSTEM SECURITY

Checking for security holes in the file system is another important part of making your system secure. Primarily, you need to check for files that can be modified by unauthorized users, files that can inadvertently grant users too many permissions, and files that can inadvertently grant access to crackers. It is also important to be able to detect unauthorized modifications to the file system, and to recover from these modifications when they are made.

3.3.1 The find Command

The *find* command [Sun88a, 183-185] is a general-purpose command for searching the file system. Using various arguments, complex matching patterns based on a file's name, type, mode, owner, modification time, and other characteristics, can be constructed. The names of files that are found using these patterns can then be printed out, or given as arguments to other UNIX commands. The general format of a *find* command is

% **find** *directories options*

where *directories* is a list of directory names to search (e.g., */usr*), and *options* contains the options to control what is being searched for. In general, for the examples in this section, you will always want to search from the root of the file system (*/*), in order to find all files matching the patterns presented.

This section describes how to use *find* to search for four possible security problems that were described in Section 2.

3.3.1.1 Finding Setuid and Setgid Files

It is important to check the system often for unauthorized setuid and setgid programs. Because these programs grant special privileges to the user who is executing them, it is necessary to ensure that insecure programs are not installed. Setuid ''root'' programs should be closely guarded – a favorite trick of many crackers is to break into ''root'' once, and leave a setuid program hidden somewhere that will enable them to regain super-user powers even if the original hole is plugged.

The command to search for setuid and setgid files is

```
# find / -type f -a \( -perm -4000 -o -perm -2000 \) -print
```

The options to this command have the following meanings:

`/` The name of the directory to be searched. In this case, we want to search the entire file system, so we specify `/`. You might instead restrict the search to */usr* or */home*.

`-type f`
Only examine files whose type is ‘‘f,’’ regular file. Other options include ‘‘d’’ for directory, ‘‘l’’ for symbolic link, ‘‘c’’ for character-special devices, and ‘‘b’’ for block-special devices.

`-a` This specifies ‘‘and.’’ Thus, we want to know about files whose type is ‘‘regular file,’’ *and* whose permissions bits match the other part of this expression.

`\( -perm -4000 -o -perm -2000 \)`
The parentheses in this part of the command are used for grouping. Thus, everything in this part of the command matches a single pattern, and is treated as the other half of the ‘‘and’’ clause described above.

`-perm -4000`
This specifies a match if the ‘‘4000’’ bit (specified as an octal number) is set in the file’s permission modes. This is the set-user-id bit.

`-o` This specifies ‘‘or.’’ Thus, we want to match if the file has the set-user-id bit *or* the set-group-id bit set.

`-perm -2000`
This specifies a match if the ‘‘2000’’ bit (specified as an octal number) is set in the file’s permission modes. This is the set-group-id bit.

`-print`
This indicates that for any file that matches the specified expression (is a regular file *and* has the setuid *or* setgid bits set in its permissions), print its name on the screen.

After executing this command (depending on how much disk space you have, it can take anywhere from 15 minutes to a couple of hours to complete), you will have a list of files that have setuid or setgid bits set on them. You should then examine each of these programs, and determine whether they should actually have these permissions. You should be especially suspicious of programs that are *not* in one of the directories (or a subdirectory) shown below.

/bin
/etc
/usr/bin
/usr/ucb
/usr/etc

One file distributed with SunOS, */usr/etc/restore*, is distributed with the setuid bit set on it, and should not be, because of a security hole. You should be sure to remove the setuid bit from this program by executing the command

```
# chmod u-s /usr/etc/restore
```

3.3.1.2 Finding World-Writable Files

World-writable files, particularly system files, can be a security hole if a cracker gains access to your system and modifies them. Additionally, world-writable directories are dangerous, since they allow a cracker to add or delete files as he wishes. The *find* command to find all world-writable files is

```
# find / -perm -2 -print
```

In this case, we do not use the *–type* option to restrict the search, since we are interested in directories and devices as well as files. The *–2* specifies the world write bit (in octal).

This list of files will be fairly long, and will include some files that *should* be world writable. You should not be concerned if terminal devices in */dev* are world writable. You should also not be concerned about line printer error log files being world writable. Finally, symbolic links may be world writable – the permissions on a symbolic link, although they exist, have no meaning.

3.3.1.3 Finding Unowned Files

Finding files that are owned by nonexistent users can often be a clue that a cracker has gained access to your system. Even if this is not the case, searching for these files gives you an opportunity to clean up files that should have been deleted at the same time the user herself was deleted. The command to find unowned files is

```
# find / -nouser -print
```

The *–nouser* option matches files that are owned by a user id not contained in the */etc/passwd* database. A similar option, *–nogroup*, matches files owned by nonexistent groups. To find all files owned by nonexistent users *or* groups, you would use the *–o* option as follows:

```
# find / -nouser -o -nogroup -print
```

3.3.1.4 Finding .rhosts Files

As mentioned in Section 2.2.1.2, users should be prohibited from having *.rhosts* files in their accounts. To search for this, it is only necessary to search the parts of the file system that contain home directories (i.e., you can skip */* and */usr*):

```
# find /home -name .rhosts -print
```

The *–name* option indicates that the complete name of any file whose name matches *.rhosts* should be printed on the screen.

3.3.2 Checklists

Checklists can be a useful tool for discovering unauthorized changes made to system directories. They aren’t practical on file systems that contain users’ home directories since these change all the time. A checklist is a listing of all the files contained in a group of directories: their sizes, owners, modification dates, and so on. Periodically, this information is collected and compared with the information in the master checklist. Files that do not match in all attributes can be suspected of having been changed.

There are several utilities that implement checklists available from public software sites (see Section 4). However, a simple utility can be constructed using only the standard UNIX *ls* and *diff* commands.

First, use the *ls* command [Sun88a, 285] to generate a master list. This is best done immediately after installing the operating system, but can be done at any time provided you’re confident about the correctness of the files on the disk. A sample command is shown below.

```
# ls -aslgR /bin /etc /usr > MasterChecklist
```

The file *MasterChecklist* now contains a complete list of all the files in these directories. You will probably want to edit it and delete the lines for files you know will be changing often (e.g., */etc/utmp*, */usr/adm/acct*). The *MasterChecklist* file should be stored somewhere safe where a cracker is unlikely to find it (since he could otherwise just change the data in it): either on a different computer system, or on magnetic tape.

To search for changes in the file system, run the above *ls* command again, saving the output in some other file, say *CurrentList*. Now use the *diff* command [Sun88a, 150] to compare the two files:

```
# diff MasterChecklist CurrentList
```

Lines that are only in the master checklist will be printed preceded by a ‘‘<,’’ and lines that are only in the current list will be preceded by a ‘‘>.’’ If there is one line for a file, preceded by a ‘‘<,’’ this means that the file has been deleted since the master checklist was created. If there is one line for a file, preceded by a ‘‘>,’’ this means that the file has been created since the master checklist was created. If there are two lines for a single file, one preceded by ‘‘<’’ and the other by ‘‘>,’’ this indicates that some attribute of the file has changed since the master checklist was created.

By carefully constructing the master checklist, and by remembering to update it periodically (you can replace it with a copy of *CurrentList*, once you’re sure the differences between the lists are harmless), you can easily monitor your system for unauthorized changes. The software packages available from the public software distribution sites implement basically the same scheme as the one here, but offer many more options for controlling what is examined and reported.

3.3.3 Backups

It is impossible to overemphasize the need for a good backup strategy. File system backups not only protect you in the even of hardware failure or accidental deletions, but they also protect you against unauthorized file system changes made by a cracker.

A good backup strategy will dump the entire system at level zero (a ‘‘full’’ dump) at least once a month. Partial (or ‘‘incremental’’) dumps should be done at least twice a week, and ideally they should be done daily. The *dump* command [Sun88a, 1612-1614] is recommended over other programs such as *tar* and *cpio*. This is because only *dump* is capable of creating a backup that can be used to restore a disk to the exact state it was in when it was dumped. The other programs do not take into account files deleted or renamed between dumps, and do not handle some specialized database files properly.

3.4 KNOW YOUR SYSTEM

Aside from running large monitoring programs such as those described in the previous sections, simple everyday UNIX commands can also be useful for spotting security violations. By running these commands often, whenever you have a free minute (for example, while waiting for someone to answer the phone), you will become used to seeing a specific pattern of output. By being familiar with the processes normally running on your system, the times different users typically log in, and so on, you can easily detect when something is out of the ordinary.

3.4.1 The ps Command

The *ps* command [Sun88a, 399-402] displays a list of the processes running on your system. *Ps* has numerous options, too many to list here. Generally, however, for the purpose of monitoring, the option string *–alxww* is the most useful.* On a Sun system running SunOS 4.0, you should expect to see at least the following:

swapper, pagedaemon
: System programs that help the virtual memory system.

/sbin/init
: The *init* process, which is responsible for numerous tasks, including bringing up login processes on terminals.

portmap, ypbind, ypserv
: Parts of the Yellow Pages system.

biod, nfsd, rpc.mountd, rpc.quotad, rpc.lockd
: Parts of the Network File System (NFS). If the system you are looking at is not a file server, the *nfsd* processes probably won’t exist.

---

* This is true for Berkeley-based systems. On System V systems, the option string *–elf* should be used instead.

rarpd, *rpc.bootparamd*
Part of the system that allows diskless clients to boot.

Other commands you should expect to see are *update* (file system updater); *getty* (one per terminal and one for the console); *lpd* (line printer daemon); *inetd* (Internet daemon, for starting other network servers); *sh* and *csh* (the Bourne shell and C shell, one or more per logged in user). In addition, if there are users logged in, you'll probably see invocations of various compilers, text editors, and word processing programs.

3.4.2 The who and w Commands

The *who* command, as mentioned previously, displays the list of users currently logged in on the system. By running this periodically, you can learn at what times during the day various users log in. Then, when you see someone logged in at a different time, you can investigate and make sure that it's legitimate.

The *w* command [Sun88a, 588] is somewhat of a cross between *who* and *ps*. Not only does it show a list of who is presently logged in, but it also displays how long they have been idle (gone without typing anything), and what command they are currently running.

3.4.3 The ls Command

Simple as its function is, *ls* is actually very useful for detecting file system problems. Periodically, you should use *ls* on the various system directories, checking for files that shouldn't be there. Most of the time, these files will have just ''landed'' somewhere by accident. However, by keeping a close watch on things, you will be able to detect a cracker long before you might have otherwise.

When using *ls* to check for oddities, be sure to use the *–a* option, which lists files whose names begin with a period (.). Be particularly alert for files or directories named ''...'', or ''..(space)'', which many crackers like to use. (Of course, remember that ''.'' and ''..'' are supposed to be there.)

3.5 KEEP YOUR EYES OPEN

Monitoring for security breaches is every bit as important as preventing them in the first place. Because it's virtually impossible to make a system totally secure, there is always the chance, no matter how small, that a cracker will be able to gain access. Only by monitoring can this be detected and remedied.

SECTION 4
SOFTWARE FOR IMPROVING SECURITY

Because security is of great concern to many sites, a wealth of software has been developed for improving the security of UNIX systems. Much of this software has been developed at universities and other public institutions, and is available free for the asking. This section describes how this software can be obtained, and mentions some of the more important programs available.

4.1 OBTAINING FIXES AND NEW VERSIONS

Several sites on the Internet maintain large repositories of public-domain and freely distributable software, and make this material available for anonymous FTP. This section describes some of the larger repositories.

4.1.1 Sun Fixes on UUNET

Sun Microsystems has contracted with UUNET Communications Services, Inc. to make fixes for bugs in Sun software available via anonymous FTP. You can access these fixes by using the *ftp* command [Sun88a, 195-201] to connect to the host *ftp.uu.net*. Then change into the directory *sun-fixes*, and obtain a directory listing, as shown in the example on the following page.

```
% ftp ftp.uu.net
Connected to uunet.UU.NET.
220 uunet FTP server (Version 5.93 Tue Mar 20 11:01:52 EST 1990) ready.
Name (ftp.uu.net:davy): anonymous
331 Guest login ok, send ident as password.
Password:                enter your mail address yourname@yourhost here
230 Guest login ok, access restrictions apply.
ftp> cd sun-fixes
250 CWD command successful.
ftp> dir
200 PORT command successful.
150 Opening ASCII mode data connection for /bin/ls.
total 2258
-rw-r--r--  1 38       22           4558 Aug 31  1989 README
-rw-r--r--  1 38       22         484687 Dec 14  1988 ddn.tar.Z
-rw-r--r--  1 38       22         140124 Jan 13  1989 gated.sun3.Z
-rwxr-xr-x  1 38       22          22646 Dec 14  1988 in.ftpd.sun3.Z
.....
.....
-rw-r--r--  1 38       22          72119 Aug 31  1989 sendmail.sun3.Z
-rwxr-xr-x  1 38       22          99147 Aug 31  1989 sendmail.sun4.Z
-rw-r--r--  1 38       22           3673 Jul 11  1989 wall.sun3.Z
-rw-r--r--  1 38       22           4099 Jul 11  1989 wall.sun4.Z
-rwxr-xr-x  1 38       22           7955 Jan 18  1989 ypbind.sun3.Z
-rwxr-xr-x  1 38       22           9237 Jan 18  1989 ypbind.sun4.Z
226 Transfer complete.
1694 bytes received in 0.39 seconds (4.2 Kbytes/s)
ftp> quit
221 Goodbye.
%
```

The file *README* contains a brief description of what each file in this directory contains, and what is required to install the fix.

4.1.2 Berkeley Fixes

The University of California at Berkeley also makes fixes available via anonymous FTP; these fixes pertain primarily to the current release of BSD UNIX (currently release 4.3). However, even if you are not running their software, these fixes are still important, since many vendors (Sun, DEC, Sequent , etc.) base their software on the Berkeley releases.

The Berkeley fixes are available for anonymous FTP from the host *ucbarpa.berkeley.edu* in the directory *4.3/ucb-fixes* . The file *INDEX* in this directory describes what each file contains.

Berkeley also distributes new versions of *sendmail* and *named* [Sun88a, 1758-1760, 1691-1692] from this machine. New versions of these commands are stored in the *4.3* directory, usually in the files *sendmail.tar.Z* and *bind.tar.Z* , respectively.

4.1.3 Simtel-20 and UUNET

The two largest general-purpose software repositories on the Internet are the hosts *wsmr-simtel20.army.mil* and *ftp.uu.net*.

wsmr-simtel20.army.mil is a TOPS-20 machine operated by the U. S. Army at White Sands Missile Range, New Mexico. The directory *pd2:<unix-c>* contains a large amount of UNIX software, primarily taken from the *comp.sources* newsgroups. The file *000-master-index.txt* contains a master list and description of each piece of software available in the repository. The file *000-intro-unix-sw.txt* contains information on the mailing list used to announce new software, and describes the procedures used for transferring files from the archive with FTP.

ftp.uu.net is operated by UUNET Communications Services, Inc. in Falls Church, Virginia. This company sells Internet and USENET access to sites all over the country (and internationally). The software posted to the following USENET source newsgroups is stored here, in directories of the same name:

comp.sources.games
comp.sources.misc
comp.sources.sun
comp.sources.unix
comp.sources.x

Numerous other distributions, such as all the freely distributable Berkeley UNIX source code, Internet Request for Comments (RFCs), and so on are also stored on this machine.

4.1.4 Vendors

Many vendors make fixes for bugs in their software available electronically, either via mailing lists or via anonymous FTP. You should contact your vendor to find out if they offer this service, and if so, how to access it. Some vendors that offer these services include Sun Microsystems (see above), Digital Equipment Corp., the University of California at Berkeley (see above), and Apple Computer.

4.2 THE NPASSWD COMMAND

The *npasswd* command, developed by Clyde Hoover at the University of Texas at Austin, is intended to be a replacement for the standard UNIX *passwd* command [Sun88a, 379], as well as the Sun *yppasswd* command [Sun88a, 611]. *npasswd* makes passwords more secure by refusing to allow users to select insecure passwords. The following capabilities are provided by *npasswd*:

- Configurable minimum password length
- Configurable to force users to use mixed case or digits and punctuation
- Checking for ‘‘simple’’ passwords such as a repeated letter

- Checking against the host name and other host-specific information
- Checking against the login name, first and last names, and so on
- Checking for words in various dictionaries, including the system dictionary.

The *npasswd* distribution is available for anonymous FTP from *emx.utexas.edu* in the directory *pub/npasswd*.

4.3 THE COPS PACKAGE

COPS is a security tool for system administrators that checks for numerous common security problems on UNIX systems, including many of the things described in this document. COPS is a collection of shell scripts and C programs that can easily be run on almost any UNIX variant. Among other things, it checks the following items and sends the results to the system administrator:

- Checks */dev/kmem* and other devices for world read/writability.
- Checks special/important files and directories for ‘‘bad’’ modes (world writable, etc.).
- Checks for easily guessed passwords.
- Checks for duplicate user ids, invalid fields in the password file, etc.
- Checks for duplicate group ids, invalid fields in the group file, etc.
- Checks all users’ home directories and their *.cshrc*, *.login*, *.profile*, and *.rhosts* files for security problems.
- Checks all commands in the */etc/rc* files [Sun88a, 1724-1725] and *cron* files [Sun88a, 1606-1607] for world writability.
- Checks for bad ‘‘root’’ paths, NFS file system exported to the world, etc.
- Includes an expert system that checks to see if a given user (usually ‘‘root’’) can be compromised, given that certain rules are true.
- Checks for *changes* in the setuid status of programs on the system.

The COPS package is available from the *comp.sources.unix* archive on *ftp.uu.net*, and also from the repository on *wsmr-simtel20.army.mil*.

4.4 SUN C2 SECURITY FEATURES

With the release of SunOS 4.0, Sun has included security features that allow the system to operate at a higher level of security, patterned after the C2* classification. These features can be installed as one of the options when installing the system from the distribution tapes. The

* C2 is one of several security classifications defined by the National Computer Security Center, and is described in [NCSC85], the ‘‘orange book.’’

security features added by this option include

- Audit trails that record all login and logout times, the execution of administrative commands, and the execution of privileged (setuid) operations.
- A more secure password file mechanism (‘‘shadow password file’’) that prevents crackers from obtaining a list of the encrypted passwords.
- DES encryption capability.
- A (more) secure NFS implementation that uses public-key encryption to authenticate the users of the system and the hosts on the network, to be sure they really are who they claim to be.

These security features are described in detail in [Sun88c].

4.5 KERBEROS

Kerberos [Stei88] is an authentication system developed by the Athena Project at the Massachusetts Institute of Technology. Kerberos is a third-party authentication service, which is trusted by other network services. When a user logs in, Kerberos authenticates that user (using a password), and provides the user with a way to prove her identity to other servers and hosts scattered around the network.

This authentication is then used by programs such as *rlogin* [Sun88a, 418-419] to allow the user to log in to other hosts without a password (in place of the *.rhosts* file). The authentication is also used by the mail system in order to guarantee that mail is delivered to the correct person, as well as to guarantee that the sender is who he claims to be. NFS has also been modified by M.I.T. to work with Kerberos, thereby making the system much more secure.

The overall effect of installing Kerberos and the numerous other programs that go with it is to virtually eliminate the ability of users to ‘‘spoof’’ the system into believing they are someone else. Unfortunately, installing Kerberos is very intrusive, requiring the modification or replacement of numerous standard programs. For this reason, a source license is usually necessary. There are plans to make Kerberos a part of 4.4BSD, to be released by the University of California at Berkeley sometime in 1990.

SECTION 5
KEEPING ABREAST OF THE BUGS

One of the hardest things about keeping a system secure is finding out about the security holes before a cracker does. To combat this, there are several sources of information you can and should make use of on a regular basis.

5.1 THE COMPUTER EMERGENCY RESPONSE TEAM

The Computer Emergency Response Team (CERT) was established in December 1988 by the Defense Advanced Research Projects Agency to address computer security concerns of research users of the Internet. It is operated by the Software Engineering Institute at Carnegie-Mellon University. The CERT serves as a focal point for the reporting of security violations, and the dissemination of security advisories to the Internet community. In addition, the team works with vendors of various systems in order to coordinate the fixes for security problems.

The CERT sends out security advisories to the *cert-advisory* mailing list whenever appropriate. They also operate a 24-hour hotline that can be called to report security problems (e.g., someone breaking into your system), as well as to obtain current (and accurate) information about rumored security problems.

To join the *cert-advisory* mailing list, send a message to *cert@cert.sei.cmu.edu* and ask to be added to the mailing list. Past advisories are available for anonymous FTP from the host *cert.sei.cmu.edu* . The 24-hour hotline number is (412) 268-7090.

5.2 DDN MANAGEMENT BULLETINS

The *DDN Management Bulletin* is distributed electronically by the Defense Data Network (DDN) Network Information Center under contract to the Defense Communications Agency. It is a means of communicating official policy, procedures, and other information of concern to management personnel at DDN facilities.

The *DDN Security Bulletin* is distributed electronically by the DDN SCC (Security Coordination Center), also under contract to DCA, as a means of communicating information on network and host security exposures, fixes, and concerns to security and management personnel at DDN facilities.

Anyone may join the mailing lists for these two bulletins by sending a message to *nic@nic.ddn.mil* and asking to be placed on the mailing lists.

5.3 SECURITY-RELATED MAILING LISTS

There are several other mailing lists operated on the Internet that pertain directly or indirectly to various security issues. Some of the more useful ones are described below.

5.3.1 Security

The UNIX Security mailing list exists to notify system administrators of security problems *before* they become common knowledge, and to provide security enhancement information. It is a restricted-access list, open only to people who can be verified as being principal systems people at a site. Requests to join the list must be sent by either the site contact listed in the Network Information Center's WHOIS database, or from the ''root'' account on one of the major site machines. You must include the destination address you want on the list, an indication of whether you want to be on the mail reflector list or receive weekly digests, the electronic mail address and voice telephone number of the site contact if it isn't you, and the name, address, and telephone number of your organization. This information should be sent to *security-request@cpd.com*.

5.3.2 RISKS

The RISKS digest is a component of the ACM Committee on Computers and Public Policy, moderated by Peter G. Neumann. It is a discussion forum on risks to the public in computers and related systems, and along with discussing computer security and privacy issues, has discussed such subjects as the Stark incident, the shooting down of the Iranian airliner in the Persian Gulf (as it relates to the computerized weapons systems), problems in air and railroad traffic control systems, software engineering, and so on. To join the mailing list, send a message to *risks-request@csl.sri.com*. This list is also available in the USENET newsgroup *comp.risks*.

5.3.3 TCP-IP

The TCP-IP list is intended to act as a discussion forum for developers and maintainers of implementations of the TCP/IP protocol suite. It also discusses network-related security problems when they involve programs providing network services, such as *sendmail*. To join the TCP-IP list, send a message to *tcp-ip-request@nic.ddn.mil*. This list is also available in the USENET newsgroup *comp.protocols.tcp-ip*.

5.3.4 SUN-SPOTS, SUN-NETS, SUN-MANAGERS

The SUN-SPOTS, SUN-NETS, and SUN-MANAGERS lists are all discussion groups for users and administrators of systems supplied by Sun Microsystems. SUN-SPOTS is a fairly general list, discussing everything from hardware configurations to simple UNIX questions. To subscribe, send a message to *sun-spots-request@rice.edu* . This list is also available in the USENET newsgroup *comp.sys.sun* .

SUN-NETS is a discussion list for items pertaining to networking on Sun systems. Much of the discussion is related to NFS, Yellow Pages, and name servers. To subscribe, send a message to *sun-nets-request@umiacs.umd.edu* .

SUN-MANAGERS is a discussion list for Sun system administrators and covers all aspects of Sun system administration. To subscribe, send a message to *sun-managers-request@eecs.nwu.edu* .

5.3.5 VIRUS-L

The VIRUS-L list is a forum for the discussion of computer virus experiences, protection software, and related topics. The list is open to the public, and is implemented as a mail reflector, not a digest. Most of the information is related to personal computers, although some of it may be applicable to larger systems. To subscribe, send the line

`SUB VIRUS-L` *your full name*

to the address *listserv%lehiibm1.bitnet@mitvma.mit.edu* .

SECTION 6
SUGGESTED READING

This section suggests some alternate sources of information pertaining to the security and administration of the UNIX operating system.

UNIX System Administration Handbook
Evi Nemeth, Garth Snyder, Scott Seebass
Prentice Hall, 1989, $26.95

> This is perhaps the best general-purpose book on UNIX system administration currently on the market. It covers Berkeley UNIX, SunOS, and System V. The 26 chapters and 17 appendices cover numerous topics, including booting and shutting down the system, the file system, configuring the kernel, adding a disk, the line printer spooling system, Berkeley networking, *sendmail*, and *uucp*. Of particular interest are the chapters on running as the super-user, backups, and security.

UNIX Operating System Security
F. T. Grammp and R. H. Morris
AT&T Bell Laboratories Technical Journal
October 1984

> This is an excellent discussion of some of the more common security problems in UNIX and how to avoid them, written by two of Bell Labs' most prominent security experts.

Password Security: A Case History
Robert Morris and Ken Thompson
Communications of the ACM
November 1979

> An excellent discussion on the problem of password security, and some interesting information on how easy it is to crack passwords and why. This document is usually reprinted in most vendors' UNIX documentation.

On the Security of UNIX
Dennis M. Ritchie
May 1975

> A discussion on UNIX security from one of the original creators of the system. This document is usually reprinted in most vendors' UNIX documentation.

The Cuckoo's Egg
Clifford Stoll
Doubleday, 1989, $19.95

> An excellent story of Stoll's experiences tracking down the German crackers who were breaking into his systems and selling the data they found to the KGB. Written at a level

that nontechnical users can easily understand.

System and Network Administration
Sun Microsystems
May, 1988

Part of the SunOS documentation, this manual covers most aspects of Sun system administration, including security issues. A must for anyone operating a Sun system, and a pretty good reference for other UNIX systems as well.

Security Problems in the TCP/IP Protocol Suite
S. M. Bellovin
ACM Computer Communications Review
April, 1989

An interesting discussion of some of the security problems with the protocols in use on the Internet and elsewhere. Most of these problems are far beyond the capabilities of the average cracker, but it is still important to be aware of them. This article is technical in nature, and assumes familiarity with the protocols.

A Weakness in the 4.2BSD UNIX TCP/IP Software
Robert T. Morris
AT&T Bell Labs Computer Science Technical Report 117
February, 1985

An interesting article from the author of the Internet worm, which describes a method that allows remote hosts to ‘‘spoof’’ a host into believing they are trusted. Again, this article is technical in nature, and assumes familiarity with the protocols.

Computer Viruses and Related Threats: A Management Guide
John P. Wack and Lisa J. Carnahan
National Institute of Standards and Technology
Special Publication 500-166

This document provides a good introduction to viruses, worms, trojan horses, and so on, and explains how they work and how they are used to attack computer systems. Written for the nontechnical user, this is a good starting point for learning about these security problems. This document can be ordered for $2.50 from the U. S. Government Printing Office, document number 003-003-02955-6.

SECTION 7
CONCLUSIONS

Computer security is playing an increasingly important role in our lives as more and more operations become computerized, and as computer networks become more widespread. In order to protect your systems from snooping and vandalism by unauthorized crackers, it is necessary to enable the numerous security features provided by the UNIX system.

In this document, we have covered the major areas that can be made more secure:

- Account security
- Network security
- File system security.

Additionally, we have discussed how to monitor for security violations, where to obtain security-related software and bug fixes, and numerous mailing lists for finding out about security problems that have been discovered.

Many crackers are not interested in breaking into specific systems, but rather will break into any system that is vulnerable to the attacks they know. Eliminating these well-known holes and monitoring the system for other security problems will usually serve as adequate defense against all but the most determined crackers. By using the procedures and sources described in this document, you *can* make your system more secure.

REFERENCES

[Eich89] Eichin, Mark W., and Jon A. Rochlis. *With Microscope and Tweezers: An Analysis of the Internet Virus of November 1988*. Massachusetts Institute of Technology. February 1989.

[Elme88] Elmer-DeWitt, Philip. ‘‘ ‘The Kid Put Us Out of Action.’ ’’ *Time*, 132 (20): 76, November 14, 1988.

[Gram84] Grammp, F. T., and R. H. Morris. ‘‘UNIX Operating System Security.’’ *AT&T Bell Laboratories Technical Journal*, 63 (8): 1649-1672, October 1984.

[Hind83] Hinden, R., J. Haverty, and A. Sheltzer. ‘‘The DARPA Internet: Interconnecting Heterogeneous Computer Networks with Gateways.’’ *IEEE Computer Magazine*, 16 (9): 33-48, September 1983.

[McLe87] McLellan, Vin. ‘‘NASA Hackers: There’s More to the Story.’’ *Digital Review*, November 23, 1987, p. 80.

[Morr78] Morris, Robert, and Ken Thompson. ‘‘Password Security: A Case History.’’ *Communications of the ACM*, 22 (11): 594-597, November 1979. Reprinted in *UNIX System Manager’s Manual*, 4.3 Berkeley Software Distribution. University of California, Berkeley. April 1986.

[NCSC85] National Computer Security Center. *Department of Defense Trusted Computer System Evaluation Criteria*, Department of Defense Standard DOD 5200.28-STD, December, 1985.

[Quar86] Quarterman, J. S., and J. C. Hoskins. ‘‘Notable Computer Networks.’’ *Communications of the ACM*, 29 (10): 932-971, October 1986.

[Reed84] Reeds, J. A., and P. J. Weinberger. ‘‘File Security and the UNIX System Crypt Command.’’ *AT&T Bell Laboratories Technical Journal*, 63 (8): 1673-1683, October 1984.

[Risk87] *Forum on Risks to the Public in Computers and Related Systems*. ACM Committee on Computers and Public Policy, Peter G. Neumann, Moderator. Internet mailing list. Issue 5.73, December 13, 1987.

[Risk88] *Forum on Risks to the Public in Computers and Related Systems*. ACM Committee on Computers and Public Policy, Peter G. Neumann, Moderator. Internet mailing list. Issue 7.85, December 1, 1988.

[Risk89a] *Forum on Risks to the Public in Computers and Related Systems*. ACM Committee on Computers and Public Policy, Peter G. Neumann, Moderator. Internet mailing list. Issue 8.2, January 4, 1989.

[Risk89b] *Forum on Risks to the Public in Computers and Related Systems*. ACM Committee on Computers and Public Policy, Peter G. Neumann, Moderator. Internet mailing list. Issue 8.9, January 17, 1989.

[Risk90] *Forum on Risks to the Public in Computers and Related Systems*. ACM Committee on Computers and Public Policy, Peter G. Neumann, Moderator. Internet mailing list. Issue 9.69, February 20, 1990.

[Ritc75] Ritchie, Dennis M. ''On the Security of UNIX.'' May 1975. Reprinted in *UNIX System Manager's Manual*, 4.3 Berkeley Software Distribution. University of California, Berkeley. April 1986.

[Schu90] Schuman, Evan. ''Bid to Unhook Worm.'' *UNIX Today!*, February 5, 1990, p. 1.

[Seel88] Seeley, Donn. *A Tour of the Worm*. Department of Computer Science, University of Utah. December 1988.

[Spaf88] Spafford, Eugene H. *The Internet Worm Program: An Analysis*. Technical Report CSD-TR-823. Department of Computer Science, Purdue University. November 1988.

[Stee88] Steele, Guy L. Jr., Donald R. Woods, Raphael A. Finkel, Mark R. Crispin, Richard M. Stallman, and Geoffrey S. Goodfellow. *The Hacker's Dictionary*. New York: Harper and Row, 1988.

[Stei88] Stein, Jennifer G., Clifford Neuman, and Jeffrey L. Schiller. ''Kerberos: An Authentication Service for Open Network Systems.'' *USENIX Conference Proceedings*, Dallas, Texas, Winter 1988, pp. 203-211.

[Stol88] Stoll, Clifford. ''Stalking the Wily Hacker.'' *Communications of the ACM*, 31 (5): 484-497, May 1988.

[Stol89] Stoll, Clifford. *The Cuckoo's Egg*. New York: Doubleday, 1989.

[Sun88a] Sun Microsystems. *SunOS Reference Manual*, Part Number 800-1751-10, May 1988.

[Sun88b] Sun Microsystems. *System and Network Administration*, Part Number 800-1733-10, May 1988.

[Sun88c] Sun Microsystems. *Security Features Guide*, Part Number 800-1735-10, May 1988.

[Sun88d] Sun Microsystems. ''Network File System: Version 2 Protocol Specification.'' *Network Programming*, Part Number 800-1779-10, May 1988, pp. 165-185.

APPENDIX A – SECURITY CHECKLIST

This checklist summarizes the information presented in this paper, and can be used to verify that you have implemented everything described.

Account Security

- ☐ Password policy developed and distributed to all users
- ☐ All passwords checked against obvious choices
- ☐ Expiration dates on all accounts
- ☐ No ‘‘idle’’ guest accounts
- ☐ All accounts have passwords or ‘‘*’’ in the password field
- ☐ No group accounts
- ☐ ‘‘+’’ lines in *passwd* and *group* checked if running Yellow Pages

Network Security

- ☐ *hosts.equiv* contains only local hosts, and no ‘‘+’’
- ☐ No *.rhosts* files in users’ home directories
- ☐ Only local hosts in ‘‘root’’ *.rhosts* file, if any
- ☐ Only ‘‘console’’ labeled as ‘‘secure’’ in *ttytab* (servers only)
- ☐ No terminals labeled as ‘‘secure’’ in *ttytab* (clients only)
- ☐ No NFS file systems exported to the world
- ☐ *ftpd* version later than December, 1988
- ☐ No ‘‘decode’’ alias in the aliases file
- ☐ No ‘‘wizard’’ password in *sendmail.cf*
- ☐ No ‘‘debug’’ command in *sendmail*
- ☐ *fingerd* version later than November 5, 1988
- ☐ Modems and terminal servers handle hangups correctly

File System Security

- ☐ No setuid or setgid shell scripts
- ☐ Check all ‘‘nonstandard’’ setuid and setgid programs for security
- ☐ Setuid bit removed from */usr/etc/restore*
- ☐ Sticky bits set on world-writable directories
- ☐ Proper umask value on ‘‘root’’ account
- ☐ Proper modes on devices in */dev*

Backups

- ☐ Level 0 dumps at least monthly
- ☐ Incremental dumps at least bi-weekly

This page intentionally left blank.
Just throw it out.

CONTENTS

1 INTRODUCTION 1
1.1 UNIX Security 1
1.2 The Internet Worm 2
1.3 Spies and Espionage 2
1.4 Other Break-Ins 3
1.5 Security is Important 3

2 IMPROVING SECURITY 5
2.1 Account Security 5
2.1.1 Passwords 5
2.1.1.1 Selecting Passwords 6
2.1.1.2 Password Policies 7
2.1.1.3 Checking Password Security 7
2.1.2 Expiration Dates 8
2.1.3 Guest Accounts 8
2.1.4 Accounts Without Passwords 9
2.1.5 Group Accounts and Groups 9
2.1.6 Yellow Pages 10
2.2 Network Security 11
2.2.1 Trusted Hosts 11
2.2.1.1 The hosts.equiv File 11
2.2.1.2 The .rhosts File 12
2.2.2 Secure Terminals 12
2.2.3 The Network File System 13
2.2.3.1 The exports File 13
2.2.3.2 The netgroup File 14
2.2.3.3 Restricting Super-User Access 16
2.2.4 FTP 16
2.2.4.1 Trivial FTP 17
2.2.5 Mail 18
2.2.6 Finger 19
2.2.7 Modems and Terminal Servers 19
2.2.8 Firewalls 20
2.3 File System Security 20
2.3.1 Setuid Shell Scripts 21
2.3.2 The Sticky Bit on Directories 22
2.3.3 The Setgid Bit on Directories 22
2.3.4 The umask Value 22
2.3.5 Encrypting Files 23
2.3.6 Devices 23
2.4 Security Is Your Responsibility 24

CONTENTS (continued)

3 MONITORING SECURITY....25
3.1 Account Security....25
3.1.1 The lastlog File....25
3.1.2 The utmp and wtmp Files....25
3.1.3 The acct File....26
3.2 Network Security....27
3.2.1 The syslog Facility....27
3.2.2 The showmount Command....28
3.3 File System Security....29
3.3.1 The find Command....29
3.3.1.1 Finding Setuid and Setgid Files....29
3.3.1.2 Finding World-Writable Files....31
3.3.1.3 Finding Unowned Files....31
3.3.1.4 Finding .rhosts Files....31
3.3.2 Checklists....32
3.3.3 Backups....33
3.4 Know Your System....33
3.4.1 The ps Command....33
3.4.2 The who and w Commands....34
3.4.3 The ls Command....34
3.5 Keep Your Eyes Open....34

4 SOFTWARE FOR IMPROVING SECURITY....35
4.1 Obtaining Fixes and New Versions....35
4.1.1 Sun Fixes on UUNET....35
4.1.2 Berkeley Fixes....36
4.1.3 Simtel-20 and UUNET....37
4.1.4 Vendors....37
4.2 The npasswd Command....37
4.3 The COPS Package....38
4.4 Sun C2 Security Features....38
4.5 Kerberos....39

5 KEEPING ABREAST OF THE BUGS....41
5.1 The Computer Emergency Response Team....41
5.2 DDN Management Bulletins....41
5.3 Security-Related Mailing Lists....42
5.3.1 Security....42
5.3.2 RISKS....42
5.3.3 TCP-IP....42

CONTENTS (concluded)

5.3.4 SUN-SPOTS, SUN-NETS, SUN-MANAGERS....................42
5.3.5 VIRUS-L....................43

6 SUGGESTED READING....................45

7 CONCLUSIONS....................47

REFERENCES....................49

APPENDIX A - SECURITY CHECKLIST....................51