Unix Security

Objectives

Understand the security features provided by a typical operating system.

Introduce the basic Unix security model.

See how general security principles are implemented in an actual operating system.

This is not a crash course on Unix security administration.

Agenda

Unix security – background

Principals, subjects, objects

Access rules

Security patterns

- Controlled invocation (SUID programs)
- Securing memory and devices
- Importing data
- Finding resources

Wrappers

Managing Unix security

Unix Preliminaries

There are several flavours of Unix; each vendor makes its own enhancements.

Vendor versions differ in the way some security controls are managed enforced.

- Commands and filenames used in this lecture are indicative of typical use but may differ from actual systems.

Unix designed originally for small multi-user computers in a network environment;
later scaled up to commercial servers and down to PCs.

Unix Preliminaries

Unix (like the Internet) was developed for friendly environments like research labs or universities.

Security mechanisms were quite weak and elementary; improved gradually.

Unix implements discretionary access control with a granularity of owner, group, other.

There exist “secure” versions of Unix: Trusted Unix or Secure Unix often indicates support for multi-level security.

Unix Design Philosophy

Security managed by a skilled administrator, not by users.

Command line tools and scripting.

Archaic syntax retained; those who already know it, love it (saves keystrokes!).

Focus on:

- protecting users from each other.
- protecting against attacks from the network.

Vendor-specific solutions for large system management and user-administered PCs.

Principals

User identifiers (UIDs) and group identifiers (GIDs) are the principals.

A UID (GID) is a 16-bit number; examples:

- 0: root
- 1: bin
- 2: daemon
- 8: mail
- 9: news
- 261: diego

UID values differ from system to system

Superuser (root) UID is always zero.

User Accounts

Information about principals is stored in user accounts and home directories.

User accounts stored in the **`/etc/passwd`** file

```
% more /etc/passwd
```

User account format:

username:password:UID:GID:name:homedir:shell

Example:

```
dieter:RT.QsZEEsxT92:1026:53:Dieter
  Gollmann:/home/staff/dieter:/bin/bash
```

User Account Details

User name: up to eight characters long

Password: stored “encrypted” (really a hash)

User ID: user identifier for access control

group ID: user’s primary group

ID string: the user's full name

home directory

Login shell: the program started after successful log in

Example

Some lines from **/etc/passwd:**

```
root:7kSSI2k.Df:0:0:root:/root:/bin/bash
mail:x:8:12:mail:/var/spool/mail:
news:x:9:13:news:/var/spool/news:
ace:69geDfelkw:500:103:Alice:/home/ace:/bin/bash
carol:7fkKdefh3d:501:102:Carol:/home/carol:/bin/nologin
al:Hj9XDdw0Pi:503:102::/home/al:/bin/bash
```

Superuser

The superuser is a special privileged principal with UID 0 and usually the user name root.

There are few restrictions on the superuser:

- All security checks are turned off for superuser.
- The superuser can become any other user.
- The superuser can change the system clock.

The superuser cannot write to a read-only file system but can remount it as writeable.

The superuser cannot decrypt passwords but can reset them.

Groups

Users belong to one or more groups.

The file **`/etc/group`** contains a list of all groups; file entry format:

groupname:password:GID:list of users

Example:

```
infosecwww:*:209:carol,al
```

Every user belongs to a primary group; the group ID (GID) of the primary group is stored in **`/etc/passwd`**.

Group Membership

System V Unix:

- a user can only be in one group at a time;
- Command to change current group: **`newgrp`**
- Users can gain temporary membership in a group where they are not members if they provide the correct group password.

Berkeley Unix:

- a user can reside in more than one group; there is no need for **`newgrp`**.

Groups and Access Control

Collecting users in groups is a convenient basis for access control decisions.

For example, put all users allowed to access email in a group called **`mail`** or put all operators in a group **`operator`**.

Conflicts in user rights can occur:

- Owner is allowed, group isn't (and vice versa).
- What happens if the group has access but not the owner?

Subjects

The subjects in Unix are processes; a process has a process ID (PID).

New processes generated with **`exec`** or **`fork`**.

Processes have a real UID/GID and an effective UID/GID.

Real UID/GID: inherited from the parent; typically UID/GID of the user logged in.

Effective UID/GID: inherited from the parent process or from the file being executed.

POSIX compliant versions add saved UID/GID.

Example

Process	UID real	UID effective	GID real	GID effective
/bin/login	**root**	**root**	**system**	**system**

User dieter logs on; the login process verifies the password and changes its UID and GID:

/bin/login dieter dieter staff staff

The login process executes the user's login shell:

/bin/bash dieter dieter staff staff

From the shell, the user executes a command, e.g. **ls**

/bin/ls dieter dieter staff staff

The User executes command **su** to start a new shell as root:

/bin/bash dieter root staff system

Passwords

Users are identified by user name and authenticated by password.

Passwords stored in **`/etc/passwd`** "encrypted" with the algorithm crypt(3).

crypt(3) is really a one-way function:
a slightly modified DES algorithm repeated 25 times with the all-zero block as start value and the password as key.

Salting: password encrypted together with a 12-bit random "salt" that is stored in the clear.

Passwords

When the password field for a user is empty, the user does not need a password to log in.

To disable a user account, let the password field starts with an asterisk; applying the one-way function to a password can never result in an asterisk.

`/etc/passwd` is world-readable as many programs require data from user accounts; makes password-guessing attacks easy.

Shadow password files: passwords are not stored in **`/etc/passwd`** but in a shadow file that can only be accessed by root.

/etc/shadow

Also used for password aging and automatic account locking; file entries have nine fields:

- username
- user password
- days since password was changed
- days left before user may change password
- days left before user is forced to change password
- days to “change password” warning
- days left before password is disabled
- days since the account has been disabled
- reserved

Objects

Files, directories, memory devices, I/O devices are uniformly treated as resources.

These resources are the objects of access control.

The resources are organized in a tree-structured file system.

Each file entry in a directory is a pointer to a data structure called inode.

Inode

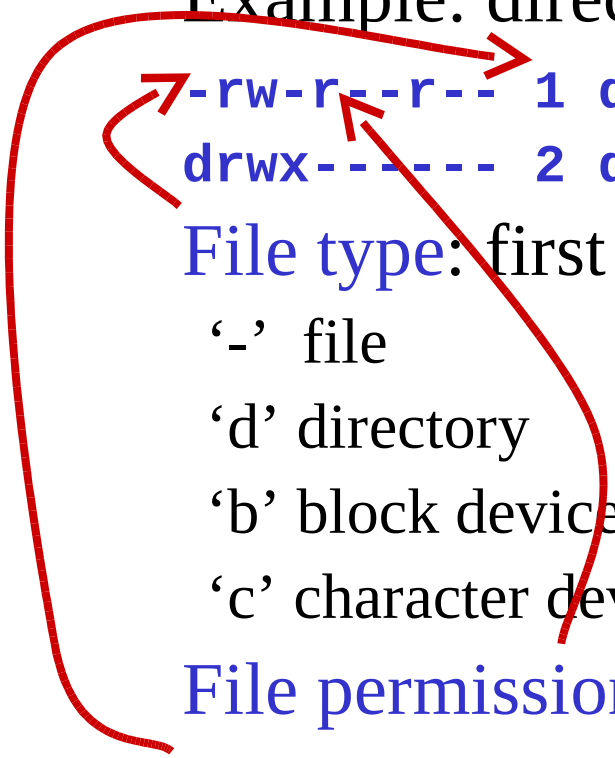
mode	type of file and access rights
uid	username of the owner
gid	owner group
atime	access time
mtime	modification time
itime	inode alteration time
block count	size of file
	physical location

Fields in the inode relevant for access control

Information about Objects

Example: directory listing with **`ls -l`**

```
-rw-r--r-- 1 dieter staff 1617 Oct 28 11:01 my.tex
drwx------ 2 dieter staff  512 Oct 25 17:44 ads/
```



File type: first character

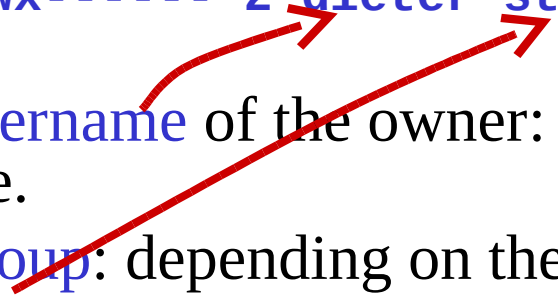
- '-' file
- 'd' directory
- 'b' block device file
- 'c' character device file
- 's' socket
- 'l' symbolic link
- 'p' FIFO

File permissions: next nine characters

Link counter: the number of links (i.e. firectory entries pointing to) the file

Information about Objects

```
-rw-r--r-- 1 dieter staff 1617 Oct 28 11:01 my.tex
drwx------ 2 dieter staff  512 Oct 25 17:44 ads/
```



Username of the owner: usually the user that has created the file.

Group: depending on the version of Unix, a newly created file belongs to its creator's group or to its directory's group.

File size, modification time, filename.

Owner and root can change permissions (**chmod**);
root can change file owner and group (**chown**).

Filename is stored in the directory, not in inode.

File Permissions

Permission bits are grouped in three triples that define read, write, and execute access for owner, group, and other.

A '-' indicates that a right is not granted.

`rw-r--r--` read and write access for the owner, read access for group and other.

`rwx------` read, write, and execute access for the owner, no rights to group and other.

Binary Representation

3-Bit for user rights

- 1 means accept, 0 means reject

Read only

R	W	E
1	0	0

Read/Write

R	W	E
1	1	0

Execute

R	W	E
1	1	1

Treat bits as binary values:

- Read only => 100_B => 4
- Read/Write => 110_B => 6
- Read/Write/Execute => 111_B => 7

Octal Representation

The three bit range is 0-7 => octal numbers are sufficient.

Examples:

- `rw-r--r--` is equivalent to 644

  Owner Read/Write; Group, Any: Read

- `rwxrwxrwx` is equivalent to 777

  Owner, Group, Any: Read/Write/Exec

Three additional bits needed for:

- Set UID to owner's (SUID).
- Set GID to owning group's (SGID).
- Sticky bit.

4 Character Octal Number

U	G	S
1	0	1

U: SUID

G: SGID

S: Sticky bit

Conversion table:

0040 read by group
0020 write by group
0010 execute by group
0004 read by other
0002 write by other
0001 execute by other

4000 set UID on execution
2000 set GID on execution
1000 set sticky bit
0400 read by owner
0200 write by owner
0100 execute by owner

File Permissions continued

When **`ls -l`** displays a SUID program, the execute permission of the owner is given as **`s`** instead of **`x`**:

```
-rws--x—x 3 root bin 16384 Nov 16 1996 passwd*
```

When **`ls -l`** displays a SGID program, the execute permission of the group is given as **`S`** instead of **`x`**:

```
-rwx--S—x 3 root bin 16384 Nov 16 1996 passwd*
```

Permissions for Directories

Every user has a home directory; to put files and subdirectories into, the correct permissions for the directory are required.

Read permission: to find which files are in the directory, e.g. for executing **`ls`**.

Write permission: to add files to and remove files from the directory.

Execute permission: to make the directory the current directory (**`cd`**) and for opening files inside the directory.

Permissions for Directories

You can open a file in the directory if you know that the file exists but you cannot use `ls` to see what is in the directory.

To access your own files, you need execute permission in the directory.

To stop other users from reading your files, you could either set the access permissions on the files or prevent access to the directory.

To delete a file, you need write and execute permission for the directory; you do not need any permission on the file itself, it can even belong to another user.

Sticky Bit

Job queues for printing etc., are often implemented as a world-writable directories; anyone can add a file.

Problem: anyone can also delete files.

Sticky bit on a directory allows only the owner of a file (and to superuser) to delete it.

chmod, chown

Access rights can be altered with **`chmod`** command:

- **`chmod 0754 filename`**
- **`chmod u+wrx,g+rx,g-w,o+r,o-wx filename`**

Ownership can be altered with the **`chown`** command:

- **`chown nOwner:nGroup filename`**

Permissions: Order of Checking

Access control uses the effective UID/GID:

- If the subject's UID owns the file, the permission bits for owner decide whether access is granted.
- If the subject's UID does not own the file but its GID does, the permission bits for group decide whether access is granted.
- If the subject's UID and GID do not own the file, the permission bits for other (also called world) decide whether access is granted.

Permission bits can give the owner less access than is given to the other users; the owner can always change the permissions.

Default Permissions

Unix utilities typically use default permissions 666 when creating a new file and permissions 777 when creating a new program.

Permissions can be further adjusted by the umask: a three-digit octal number specifying the rights that should be withheld.

The actual default permission is derived by masking the given default permissions with the umask: compute the logical AND of the bits in the default permission and of the inverse of the bits in the umask.

Default Permissions

Example: default permission 666, umask 077

Invert 077: result 700

```
 0666
 0700
------
 0600
```

The owner of the file has read and write access, all other access is denied.

umask 777 denies every access, umask 000 does not add any further restrictions .

Sensible umask Settings

022: all permissions for the owner, read and execute permission for group and other.

027: all permissions for the owner, read and execute for group and no permission for other.

037: all permissions for the owner, read permission for group, no permissions for other.

077: all permissions for the owner, no permissions for group and other.

Security Patterns

We will discuss how some general security principles manifest themselves in Unix.

Controlled invocation: SUID programs.

Physical and logical representation of objects: deleting files.

Access to the layer below: protecting devices.

Searchpath

Importing data from outside: mounting filesystems.

Controlled Invocation

Superuser privilege is required to execute certain operating system functions.

Example: only processes running as root can listen at the "trusted ports" 0 – 1023.

Solution adopted in Unix: SUID (set userID) programs and SGID (set groupID) programs.

SUID (SGID) programs run with the effective user ID or group ID of their owner or group, giving controlled access to files not normally accessible to other users.

SUID to Root

When root is the owner of a SUID program, a user executing this program will get superuser status during execution.

Important SUID programs:

`/bin/passwd`	change password
`/bin/login`	login program
`/bin/at`	batch job submission
`/bin/su`	change UID program

As the user has the program owner's privileges when running a SUID program, the program should only do what the owner intended

SUID Dangers

By tricking a SUID program owned by root to do unintended things, an attacker can act as the root.

All user input (including command line arguments and environment variables) must be processed with extreme care.

Programs should have SUID status only if it is really necessary.

The integrity of SUID programs must be monitored (tripwire).

Applying Controlled Invocation

Sensitive resources, like a web server, can be protected by combining ownership, permission bits, and SUID programs:

Create a new UID that owns the resource and all programs that need access to the resource.

Only the owner gets access permission to the resource.

Define all the programs that access the resource as SUID programs.

Managing Security

Beware of overprotection; if you deny users direct access to a file they need to perform their job, you have to provide indirect access through SUID programs.

A flawed SUID program may give users more opportunities for access than wisely chosen permission bits.

This is particularly true if the owner of the SUID program is a privileged user like root.

Deleting Files

General issue: logical vs physical memory

Unix has two ways of copying files.

`cp` creates an identical but independent copy owned by the user running **`cp`** .

`ln` creates a new filename with a pointer to the original file and increases the link counter of the original file; the new file shares its contents with the original.

If the original is deleted (with **`rm`** or **`rmdir`**) it disappears from its parent directory but the contents of the file and its copy still exist.

Deleting Files

So, users may think that they have deleted a file whereas it still exists in another directory, and they still own it.

Further issue: if a process has opened a file which then is deleted by its owner, the file remains in existence until that process closes the file.

Deleting Files

Once a file has been deleted the memory allocated to this file becomes available again.

Until these memory locations are written to again, they still contain the file's contents.

To avoid such memory residues, the file can be wiped by overwriting its contents with a pattern appropriate for the storage medium before deleting it.

But advanced file systems (e.g. defragmenter) may move files around and leave copies.

Protection of Devices

General issue: logical and physical memory

Unix treats devices like files; access to memory or to a printer is controlled like access to a file by setting permission bits.

Devices commonly found in directory **`/dev`**:

`/dev/console`	console terminal
`/dev/kmem`	kernel memory map device (image of the virtual memory)
`/dev/tty`	terminal
`/dev/hd0`	hard disk

Access to the Layer Below

Attackers can bypass the controls set on files and directories if they can get access to the memory devices holding these files.

If the read or write permission bit for other is set on a memory device, an attacker can browse through memory or modify data in memory without being affected by the permissions defined for files.

Almost all devices should therefore be unreadable and unwritable by “other”.

Example

The process status command **`ps`** displays information about memory usage and thus requires access permissions for the memory devices.

Defining **`ps`** as a SUID to root program allows **`ps`** to acquire the necessary permissions but a compromise of **`ps`** would leave an attacker with root privileges.

Better solution: let group **`mem`** own the memory devices and define **`ps`** as a SGID program.

Terminal Devices

When a user logs in, a terminal file is allocated to the user who becomes owner of the file for the session.

It is convenient to give “other” read and write permission to this file so that the user can receive messages from other parties.

This introduces vulnerabilities: other parties are now able to monitor the entire traffic to and from the terminal, potentially including the user’s password.

Terminal Devices

Other parties can send commands to the user's terminal.

For example, reprogram a function key, and have these commands executed by the unwitting user.

In some systems, intelligent terminals execute some commands automatically; an attacker can then submit commands using the privileges of another user.

Changing the Root of the Filesystem

Access control can be implemented by constraining suspect processes to a sandbox environment; access to objects outside the sandbox is prevented.

The change root command **`chroot`** restricts the available part of the filesystem:

`chroot <directory> <command>`

Changes the apparent filesystem root directory from **`/`** to directory when command executes.

Only files below the new root are thereafter accessible.

Changing the Root of the Filesystem

If you employ this strategy, make sure that user programs find all system files they need.

System files are ‘expected’ to be in directories like **`/bin`**, **`/dev`**, **`/etc`**, **`/tmp`**, or **`/usr`**

New directories of the same names have to be created under the new root and populated with the files the user will need by copying or linking to the respective files in the original directories.

Mounting Filesystems

General issue: When importing objects from another security domain into your system, access control attributes of these objects must be redefined.

The Unix filesystem is built by linking together filesystems held on different physical devices under a single root `/` with the **`mount`** command.

Remote filesystems (NFS) can be mounted from other network nodes.

Users could be allowed to mount a filesystem from their own floppy disk (**`automount`**).

Mounting Filesystems

The mounted filesystems could have dangerous settings, for example SUID to root programs sitting in an attacker's directory.

Once the filesystem has been mounted, the attacker can obtain superuser status by running such a program.

The **mount** Command

mount [-r] [-o *options*] *device directory*

-r flag specifies read-only mount.

Options:

nosuid: turns off the SUID and SGID bits on the mounted filesystem.

noexec: no binaries can be executed from the mounted filesystem.

nodev: no block or character special devices can be accessed from the filesystem.

Different versions of Unix implement different options for **mount**.

Mounting Filesystems

General issue: scoping of identifiers

NFS server trusts the client to enforce access control on the mounted filesystem.

UIDs and GIDs on two Unix systems (from different vendors) may be assigned differently.

The client may misinterpret the UID or GUID even if it tries to enforce access control.

Problem: UID and GID are local identifiers; only globally unique identifiers should be used across network.

Environment Variables

Environment variables: kept by the shell, normally used to configure the behaviour of utility programs

Inherited by default from a process' parent.

A program executing another program can set the environment variables for the program called to arbitrary values.

Danger: the invoker of setuid/setgid programs is in control of the environment variables they are given.

Usually inherited, so this also applies transitively.

Not all environment variables are documented!

Inheriting things you do not want can become a security problem.

Examples

```
PATH            # The search path for shell commands (bash)
TERM            # The terminal type (bash and csh)
DISPLAY         # X11 - the name of your display
LD_LIBRARY_PATH
                # Path to search for object and shared libraries
HOSTNAME        # Name of this UNIX host
PRINTER         # Default printer (lpr)
HOME            # The path to your home directory (bash)
PS1             # The default prompt for bash
path            # The search path for shell commands (csh)
term            # The terminal type (csh)
prompt          # The default prompt for csh
home            # The path to your home directory (csh)
```

Searchpath

General principle: execution of programs taken from a ‘wrong’ location.

Users can run a program by typing its name without specifying the full pathname that gives the location of the program within the filesystem.

The shell searches for the program following the searchpath specified by the **`PATH`** environment variable in the **`.profile`** file in the user’s home directory.

Searchpath

A typical searchpath:

```
PATH=.:\$HOME/bin:/usr/ucb:/bin:
/usr/bin:/usr/local:/usr/new: /
usr/hosts
```

Directories in the searchpath are separated by ‘:’; the first entry ‘.’ is the current directory.

When a directory is found that contains a program with the name specified, the search stops and that program will be executed.

Searchpath

To insert a Trojan horse, give it the same name as an existing program and put it in a directory that is searched before the directory containing the original program.

As a defence, call programs by their full pathname, e.g. **`/bin/su`** instead of **`su`**.

Make sure that the current directory is not in the searchpath of programs executed by root

(**`ls -a`** lists all files in your home directory, **`more .profile`** shows your profile).

Network Services (telnet, ftp)

inetd daemon listens to incoming network connections

When a connection is made, *inetd* starts the requested server program and then returns to listening for further connections.

Configuration file maps port numbers to programs

Entries in the configuration file have the format:

service type protocol waitflag userid executable command-line

Example: entry for telnet

telnet stream tcp nowait root /usr/bin/in.telnetd in.telnet

Telnet Wrapper

When *inetd* receives a request for a service, it consults the configuration file and creates a new process that runs the *executable* specified.

Name of new process changed to the name given in the *command-line* field.

Usually, the name of the *executable* and the name given in *command-line* are the same.

Telnet Wrapper

This redundancy can be used for a nice trick:

Point *inetd* daemon to a wrapper program.

Use the name of the process to remember the name of the original executable; return to this executable after running the wrapper.

Example: change configuration file entry for telnet to

```
telnet stream tcp nowait root /usr/bin/tcpd in.telnetd
```

Program executed is now the TCP wrapper executable **`/usr/bin/tcpd`**.

Telnet Wrapper

Wrapper performs access control, logging, ...

- Original application: IP address filtering.

Wrapper knows the directory it is in (**`/usr/bin`**) and its own name (*in.telnetd*) so it can call the original server program (**`/usr/bin/in.telnetd`**)

Users see no difference and receive the same service as before.

Design Principle

Add another level of indirection.

The TCP wrapper performing security controls is inserted between the *inetd* daemon and the server program.

More information on Wietse Venema's home page.

Management Issues

Brief overview of several issues relevant for managing Unix systems

Protecting the root account

Networking: trusted hosts

Auditing

Keeping up-to-date

Root Account

The root account is used by the operating system for essential tasks like login, recording the audit log, or access to I/O devices.

The root account is required for performing certain system administration tasks.

Superusers are also a major weakness of Unix; an attacker achieving superuser status effectively takes over the entire system.

Separate the duties of the systems manager; e.g. let special users like **`uucp`** or **`daemon`** deal with networking; if a special users is compromised, not all is lost.

Superuser

Systems manager should not use root as their personal account.

Change to root from a user account using **`/bin/su`**; the O/S will not refer to a version of **`su`** that has been put in some other directory.

Record all **`su`** attempts in the audit log with the user who issued the command.

`/etc/passwd` and **`/etc/group`** have to be write protected; an attacker who can edit **`/etc/passwd`** can become superuser by changing its UID to 0.

Trusted Hosts

Users from a trusted host can login without password authentication; they only need to have the same user name on both hosts.

Trusted hosts of a machine are specified in **`/etc/hosts.equiv`**.

User names must be synchronized between the hosts.

Trusted hosts of a user are specified in the **`.rhosts`** file in the user's home directory.

Trusted Hosts Limitations

When the number of hosts grows, synchronizing **`hosts.equiv`** files and user names becomes tedious.

Vendor-specific tools to distribute configuration files.

User can either access all hosts in the system or nothing; exceptions difficult to configure.

Audit Logs

`/usr/adm/lastlog` records the last time a user has logged in; displayed with **`finger`**

`/var/adm/utmp` records accounting information used by the **`who`** command.

`/var/adm/wtmp` records every time a user logs in or logs out; displayed with the **`last`** command.

`/var/adm/acct` records all executed commands; displayed with **`lastcomm`**

Keeping Up-to-date

Remember: security is a moving target.

New vulnerabilities keep being detected.

Security patches have to be developed, distributed, and installed.

Tools for assessing the security of a given systems configuration exist.

Systems managers have to keep up-to-date with current threats.

CERT advisories and the like: www.cert.org, www.sans.org, www.securityfocus.com

Summary

Unix served as a case study to see how core security primitives can be implemented.

Illustrate a number of general security issues.

Also relevant, but not covered yet: network security, software security.

For practical security, it does not suffice to have a “secure” operating system; the system also has to be managed securely.