Australian National University

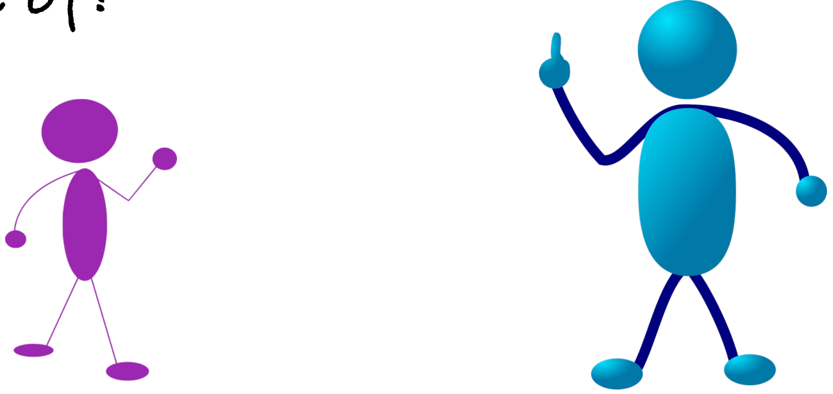
Unix Security

COMP2700 Cyber Security Foundations

Slides prepared based on Chapter 7 of Gollmann's "Comptur Security", 3rd edition.

Dad, what are clouds made of?

Linux servers, mostly.



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.

Outline

- Unix security – background
- Principals, subjects, objects
- Access rules
- Security patterns
  - Controlled invocation (SUID programs)
  - Securing memory and devices
  - Importing data
  - Finding resources
- Managing Unix security

Overview of Unix

- Unix was developed for friendly environments like research labs or universities.
- Security mechanisms were quite weak and elementary; improved gradually.
- Several flavours of Unix; 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.

Overview of Unix

- Unix designed originally for small multi-user computers in a network environment; later scaled up to commercial servers and down to PCs.
- Linux and Mac OS X are perhaps the most well-known modern Unix-like operating system.
- But lesser known, though more pervasively used, examples of Unix-like systems are (the core) of Android and iOS, running in billions of devices.

Unix Design Philosophy

- Security managed by skilled administrator, not by user. Focus on:
  - protecting users from each other.
  - protecting against attacks from the network.
- Discretionary access control with a granularity of **owner**, **group**, **other**.
- Vendor-specific solutions for managing large system and user-administered PCs.
- “Secure” versions of Unix: Trusted Unix or Secure Unix often indicates support for multi-level security.
  - E.g., Security-Enhanced Linux (SELinux) supports multi-level security.

Principals

- Principals: **user identifiers** (UIDs) and **group identifiers** (GIDs).
- A UID (GID) is a 16-bit number; examples:

    0: root

    1: bin

    2: daemon

    8: mail

    9: news

    1001: alice

- 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

```
$ cat /etc/passwd
```

- User account format:

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

User Accounts Details

- Username: up to eight characters long
- Password: password hash (in older versions of Unix); in modern Unix the password hash is stored elsewhere.
- User ID: user identifier for access control
- Group ID: user’s primary group
- ID string: user’s full name
- home directory
- Login shell: program started after successful log in

Examples

From the lab VM:

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
admin2700:x:1000:1000:Ubuntu,,,:/home/admin2700:/bin/bash
vboxadd:x:999:1::/var/run/vboxadd:/bin/false
alice:x:1001:1001:Alice,,,:/home/alice:/bin/bash
bob:x:1002:1002:Bob,,,:/home/bob:/bin/bash
charlie:x:1003:1003:Charlie,,,:/home/charlie:/bin/bash
dennis:x:1004:1004:Dennis,,,:/home/dennis:/bin/bash
eve:x:1005:1005:Eve,,,:/home/eve:/bin/bash
felix:x:1006:1006:Fong,,,:/home/fong:/bin/bash
```

Superuser

- The **superuser** is a special privileged principal with **UID 0** and usually the username **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.
- Superuser cannot write to a read-only file system but can remount it as writeable.
- Superuser cannot decrypt passwords but can reset them.

Groups

- Users belong to one or more groups.
- `/etc/group` contains all groups; file entry format:

  `groupname:password:GID:list of users`
- Every user belongs to a primary group; group ID (GID) of the primary group stored in `/etc/passwd`.
- 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`

Examples

From the lab VM: groups where user bob belongs to

```
$ cat /etc/group | grep bob
bob:x:1002:
tutors:x:1007:alice,bob,charlie
```

Examples

Some commands to display user id and groups:

```
$ whoami
alice

$ id
uid=1001(alice) gid=1001(alice)
groups=1001(alice),6(disk),1007(tutors)

$ groups
alice disk tutors
```

Sudo-ers

- In some linux distributions (such as Ubuntu), one cannot login as the root user directly.
- Instead, a special group, called ‘sudo’, is created, such that its members are allowed to become ‘root’ using the ‘sudo’ command.
- Example:

```
$ sudo whoami
root

$ grep sudo /etc/group
sudo:x:27:admin2700
```

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.

Examples

The ps command can be used to query information about processes.

For example, to display PID, real user and effective user of all processes running in the system:

```
$ ps -eo pid,ruser,euser,command
```

Example of (selected) output:

```
  PID RUSER     RUID EUSER     EUID COMMAND
 2818 alice     1001 alice     1001 bash
 3150 alice     1001 root         0 passwd
```

Passwords

- Users are identified by username and authenticated by password.
- In legacy Unix systems, passwords stored in **`/etc/passwd`** hashed with the algorithm crypt(3).
- crypt(3) is really a one-way function:
  slightly modified DES algorithm repeated 25 times with 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.
- Shadow password files: hashed passwords are not stored in `/etc/passwd` but in a shadow file **/etc/shadow** that can only be accessed by root.

Shadow password file

- Shadow password file location: /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.
- Resources organized in a tree-structured file system.
- Each file entry in a directory is a pointer to a data structure called **inode**.

Inode

Fields in the inode relevant for access control

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

Examples

The command stat displays the inode information of a file, e.g.,

```
alice@comp2700-lab:~$ stat /etc/passwd
  File: /etc/passwd
  Size: 2034      	Blocks: 8          IO Block: 4096   regular
file
Device: 811h/2065d	Inode: 8043        Links: 1
Access: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid:
(    0/    root)
Access: 2021-08-16 05:52:56.121875300 +0000
Modify: 2021-07-25 11:51:47.543481900 +0000
Change: 2021-07-25 11:51:47.583482399 +0000
 Birth: -
```

You can also use ls command to show the inode number:

```
alice@comp2700_lab:~$ ls -il /etc/passwd
8043 -rw-r--r-- 1 root root 2034 Jul 25 11:51 /etc/passwd
```

Information about Objects

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

```
-rwxr-x--- 1 alice alice 4807960 Aug 12 10:34 lab1.pdf
drwxr-xr-x 2 alice staff    4096 Aug 15 10:33 lectures
```

- File type: first character

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

- File permissions: next nine characters
- Link counter:
  - the number of links (i.e. directory entries pointing to) the file

Information about Objects

```
-rwxr-x--- 1 alice alice 4807960 Aug 17 10:34 lab1.pdf
drwxr-xr-x 2 alice tutor    4096 Aug 17 10:33 lectures
```

- 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 stored in the directory, not in inode.

File and Directory Permissions

- File permissions are internally represented by a sequence of bits, consisting of 4 groups of 3-bits.
- The first group represents *special modes* (to be discussed later).
- The next three groups define read, write, and execute access for owner, group, and other.

Australian National University

Special modes

- The first group of three bits represents special modes.
- The first bit is also called the SUID bit.
- The second bit is called the SGID bit.
- And the third is called the sticky bit.
- The SUID and SGID bits are used to implement controlled invocation (to be discussed later).
- These bits are rarely used – most files will have these bits set to 0.

Special modes

- The sticky bit is used for different purposes in different implementations.
- In some legacy Unix systems, it is used to indicate a program file should be ‘cached’ in swap space.
- In Linux, a sticky bit on a directory means that a user may not delete files owned by other users.
  - This is usually used in a world-writeable directory, such as /tmp
  - Every user can create files/directories in /tmp, but they cannot delete files/directories created by other users.

File and Directory Permissions

The three bits in the second, third and fourth groups are interpreted as follows: when the bit is set (i.e., its value is 1), its interpretation is as follows:

- First bit: read access granted
- Second bit: write access granted
- Third bit: execute access granted.

Example:

000 111 101 101

- 000 → No special modes
- 111 → Owner read,write, exec
- 101 → Group read, exec
- 101 → Other read, exec

Textual representation of permissions

- Permission bits are commonly displayed using a textual notation that is easier to understand.
- When the first group is 000 (i.e., no special modes), the remaining groups are represented textually as follows: if a bit in the group is 0, it's represented by '-'. Otherwise, depending on the position of the bit:
  - First bit: represented by 'r' (read)
  - Send bit: represented by 'w' (write)
  - Third bit: represented by 'x' (exec)
- Examples:
  - `rw-r--r--` **represents** `000 110 100 100`
  - `rwxrwxrwx` **represents** `000 111 111 111`

Special modes in textual representation

When special modes are present, the bits in the special modes change the display of the executable bits of the remaining groups.

- If SUID bit is set: display 's' if the owner exec bit is set; otherwise display 'S'.
- If SGID bit is set: display 's' if the group exec bit is set; otherwise display 'S'.
- If sticky bit is set: display 't' if the 'other' exec bit is set; otherwise display 'T'.

Special modes in textual representation

Examples:

- 110 111 110 100 can be represented as

  `rwsrwSr--`

- 011 111 101 101 can be represented as

  `rwxr-sr-t`

- 101 110 110 100 can be represented as

  `rwSrw-r-T`

Octal Representation

- Another representation of permission bits that is commonly used is the octal notation.
- Each group of three bits can be represented as an octal.
- For example:
  - 000 110 100 100 in octal notation is 0644.
  - 011 111 101 101 in octal notation is 3755.
- A 3-digit octal permissions means the special modes are absent, e.g., 644 is the same as 0644.
- Octal notations are used in some commands to set permissions ('chmod') and permission masks ('umask').

Default Permissions

- Unix utilities typically use default permissions 0666 when creating a new file and permissions 0777 when creating a new program.
- Permissions can be further adjusted by the umask:
  - a four-digit octal number specifying the rights that should be withheld.
- 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 0666, umask 0077
- Invert 0077: gives 7700, then AND:

INVERT

0077 = 000000111111

7700 = 111111000000

AND

0666 = 000110110110

7700 = 000111000000

0600 = 000110000000

- Owner of the file has read and write access, all other access is denied.
- umask 7777 denies every access, umask 0000 does not add any further restrictions .

Some umask Settings

- 0022: withhold none from owner, withhold write permission for group and for other.
- 0027: withhold none from owner, withhold write permission from group, withhold all from other.
- 0037: withhold none from owner, withhold write and execute from group, withhold all from other.
- 0077: withhold none from owner, withhold all from group and other.

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

- To access your own files, you need execute permission in the directory.
- Without read permission on the directory, but with execute permission, you can still open a file in the directory if you know that it exists but you cannot use `ls` to see what is in the directory.

Permissions for Directories

- To stop other users from reading your files, you can either set the access permissions on the files or prevent access to the directory.
- You need write and execute permission for the directory to delete a file; no permissions on the file itself are needed, it can even belong to another user.

Changing Permissions

- Access rights can be altered with **`chmod`** command:
  - **`chmod 0754 filename`**
  - **`chmod u+wrx,g+rx,g-w,o+r,o-wx filename`**
- The first octal number from the left (representing special modes) is optional, e.g.,
  - **`chmod 754 filename`**

  achieves the same thing as **`chmod 0754 filename.`**

Changing Ownership

- Ownership can be altered with the **`chown`** command:
  - **`chown <Owner>:<Group> <filename>`**

- For example:
  - **`chown alice:tutors  foo.txt`**

  changes the owner of foo.txt to user alice in group tutors.

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.
  - But the owner can always change the permissions.

Security Patterns

Some general security principles implemented in Unix.

- Controlled invocation: SUID programs.
- Physical and logical representation of objects: deleting files.
- Access to the layer below: protecting devices.
- Search path
- Importing data from outside world: 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.

Displaying SUID Programs

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

```
$ ls -l /usr/bin/passwd
-rwsr-xr-x 1 root root 59640 Mar 23  2019 /usr/bin/passwd
```

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

```
$ ls -l /usr/bin/ssh-agent
-rwxr-sr-x 1 root ssh 362640 Mar  4  2019 /usr/bin/ssh-agent
```

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/sudo`	escalate privilege to root
`/bin/su`	change UID

- 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 (confused deputy attack).
- 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 (e.g., using tripwire).

Applying Controlled Invocation

- Sensitive resources, like a web server, can be protected by combining ownership, permission bits, and SUID programs:
- **Least privilege:** 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 link counter of the original file; the new file shares its contents with the original.

– 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 random patterns before deleting it.
- But advanced file systems (e.g. defragmenter) may move files around and leave copies.

Protection of Devices

- General issue: logical vs physical memory
- In Unix, "everything is a file".
  - 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/sda1`	hard disk
`/proc`	virtual file system containing system information

Accessing 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.
  - In Linux, user group disk has write access to raw devices. Members of this group can bypass file and directory permissions.
- 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 command **`passwd`** allows any user to change their password, thus modifying the /etc/shadow file.
- Defining `passwd` as a SUID to root program allows `passwd` to acquire the necessary permissions.
- But a compromise of `passwd` would allow an attacker to modify the shadow file, e.g., to reset the administrator password.

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.
- Vulnerabilities:
  - other parties can now monitor the entire traffic to and from the terminal, potentially including the user's password.
  - Others can send commands to the user's terminal, and execute them using the privileges of another user.

Mounting File Systems

- General issue: When importing objects from another security domain into your system, access control attributes of these objects must be redefined.
- Unix file system is built by linking together file systems held on different physical devices under a single root `/` with the `mount` command.
- Remote file systems (NFS) can be mounted from other network nodes.
- Users could be allowed to mount a filesystem from their own floppy disk (`automount`).
- Mounted file systems could have dangerous settings, e.g. SUID to root programs in an attacker's directory.

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.
- Not all environment variables are documented!

Examples

The command **env** lists all the defined environment variables in the current shell.

Some 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
HOME	# The path to your home directory (bash)

Example: the “Shellshock” bug

- Discovered in September 2014.
- Exploits a vulnerability in parsing of environment variables.
- Allows an attacker to inject arbitrary codes into environment variables.
- The injected codes get executed if the target (victim) executes a bash shell.
- See

  http://en.wikipedia.org/wiki/Shellshock_(software_bug)

Search path

- 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 search path specified by the `PATH` environment variable in the `.profile` file in the user’s home directory.

Search path

- A typical search path (it may differ across different systems):

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

- Directories in the search path are separated by ':'; the first entry '.' is the current directory.
- Search paths are read from left to right.
- When a directory is found that contains a program with the name specified, the search stops and that program will be executed.

Search path

- 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/ls` instead of `ls`.
- Make sure that the current directory is not in the search path of programs executed by root.

Management Issues

- Brief overview of several issues relevant for managing Unix systems
  - Protecting the root account
  - Networking: trusted hosts
  - Auditing

Protecting the 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 a major weakness of Unix; an attacker achieving superuser status effectively takes over the entire system.
- Separate the duties of the systems manager; create users like `uucp` or `daemon` to 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

- In legacy Unix systems, commands such as rlogin or rsh allows users to login remotely.
  - Both rlogin and rsh transmit passwords in plain text
  - In modern Linux systems they are replaced by ‘secure shell’ (ssh)
- 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`**.
- Trusted hosts of a user are specified in the **`.rhosts`** file in the user’s home directory.
  - User can either access all hosts in the system or nothing; exceptions difficult to configure.

Audit Logs

In modern Linux systems, log files are located in /var/log/. For example:

- /var/log/auth.log: all authentication related events, including wrong passwords, attempts to 'sudo', etc.
- /var/log/dmesg: information related to hardware and device drivers
- /var/log/kern.log: information logged by the kernel
- /var/log/syslog: global system activity data

Audit Logs

- Audit logs may sometimes contain sensitive information.
  - Be careful of what information you log and the permissions to the log files.
- Example: bugs in Mac OS X (version 10.3.3) cause system encryption software to record disk encryption password in plaintext in installation logs.
  - See /var/log/install.log in the affected Mac OS X
  - Log accessible by normal (non-root) use. See:
    - https://www.mac4n6.com/blog/2018/3/30/omg-seriously-apfs-encrypted-plaintext-password-found-in-another-more-persistent-macos-log-file
- Example: In Android (prior to 'Jelly Bean' version), apps can request permission to read system logs.
  - See, e.g., William Enck, et. al. : A Study of Android Application Security. USENIX Security Symposium 2011

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.