ACCESS CONTROL

Privilege separation

Modern computers are

1. multi-users
2. multi-tasking

Goal: Prevent potentially misbehaving users and/or applications from harming the rest of the system

Permissions system: mechanisms for achieving separation between components

Central question

"Who is allowed to access what and how?"

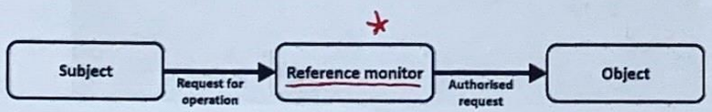
The subject (who) - *eg.* user, application, process

The object (what) - protected resource, *eg.* hardware device, network socket, memory, files, directories,

The access operation (how) - *eg.* read, write, execute

Key assumptions for separation

1. The system know who the user is - user has authenticated, *e.g.* using username / password
2. Complete mediation - all requests are mediated - all requests go to the reference monitor that enforces specified access control policies



The **reference monitor** grants permission to users to apply certain operations to a given resource

Users

Two types of accounts each with a unique identifier, the user ID (`uid`):

1. User accounts - associated with humans
2. Service accounts - associated with background processes

```
marapini@myrto-thinkpad:~$ more /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
```

You can list ALL accounts in a UNIX machine by inspecting the pswd. file that can be found in /etc/passwd

- One entry in the `/etc/passwd` per account with the fields:
  `username:password:uid:gid:uid_info:home:shell`
  (gid → group id)
- `uid 0` - user root `uid`

Groups

- Groups are sets of users that share resources
- Every group has a name and a unique identifier, the group ID (gid)
- Allow for easier users management and monitoring

You can list ALL groups by inspecting the grp. file in /etc/group

```
marapini@myrto-thinkpad:~$ more /etc/group
root:x:0:
daemon:x:1:
bin:x:2:
sys:x:3:
adm:x:4:syslog,marapini
tty:x:5:
disk:x:6:
lp:x:7:
mail:x:8:
```

There are 2 users: syslog and 'myself'

When you specify a directory, you can only specify one group

- One entry in the /etc/group per group with the fields:

  `group_name:password:gid:group_list`

File permissions

- All resources (sockets, directories, files) are managed as files
- 3 defined permissions: read (r), write (w), execute (x)
- Permissions are defined for the owner, the owner's group, and other users
- Root and owner can change file permissions
- Only root can change file ownership

Directory →

```
marapini@myrto-thinkpad:~/Documents/Work/Teaching/INFR10067-ComputerSecurity/2021/Lectures/L18.AccessControl$ ls -l
total 352
drwxrwxr-x 2 marapini marapini   4096 Feb 23 17:27 Images
-rw-r--r-- 1 marapini marapini   1839 Feb 23 17:31 L18.AccessControl.aux
-rw-r--r-- 1 marapini marapini  47121 Feb 23 17:31 L18.AccessControl.log
-rw-r--r-- 1 marapini marapini    835 Feb 23 17:31 L18.AccessControl.nav
-rw-r--r-- 1 marapini marapini      0 Feb 23 17:31 L18.AccessControl.out
-rw-r--r-- 1 marapini marapini 258111 Feb 23 17:31 L18.AccessControl.pdf
-rw-r--r-- 1 marapini marapini      0 Feb 23 17:31 L18.AccessControl.snm
-rw-rw-r-- 1 marapini marapini   9769 Feb 23 18:05 L18.AccessControl.tex
-rw-rw-r-- 1 marapini marapini  23638 Feb 20 01:28 L18.AccessControl.tex~
-rw-r--r-- 1 marapini marapini      0 Feb 23 17:31 L18.AccessControl.toc
```

(slide) pdf,

The owner (myself) has read & write permissions, NO execute permission
The rest of the group ONLY has read permission
The other users ONLY have read permission

Directory permissions

- Execute permission on a directory allows traversing it
- Read permission on a directory allows lookup

e.g. Quizz: Imagine you have the following groups:

- `infr10067` - for any user involved with the Computer Security course
- `tas` - for all Informatics TAs

TAs on INFR10067

How can you have a folder only for Computer Security TAs?

```
marapini@myrto-thinkpad:~/Documents/Work/Teaching/INFR10067-ComputerSecurity/2021/Lectures/L18.AccessControl/conjunction$ ls -l
total 4
drwxr-xr-- 3 marapini tas 4096 Feb 23 22:50 only_for_tas
marapini@myrto-thinkpad:~/Documents/Work/Teaching/INFR10067-ComputerSecurity/2021/Lectures/L18.AccessControl/conjunction$ ls -l only_for_tas/
total 4
drwxr-xr-- 2 marapini infr10067 4096 Feb 23 22:50 only_for_infr10067_tas
marapini@myrto-thinkpad:~/Documents/Work/Teaching/INFR10067-ComputerSecurity/2021/Lectures/L18.AccessControl/conjunction$
```

→ Nest 2 directories!

2 directories: only_for_tas & only_for_infr10067_tas

Inside!

In only_for_tas, I've given execute permission only to the grp tas.

In only_for_infr10067_tas, I've given execute permission only to the users in infr10067

Processes

- Each process has a unique identifier, the process ID (pid)
- Each process is associated with the user that spanned it

The init process which is run by the root user is given pid of 0

```
marapini@myrto-thinkpad:~$ ps -ef
UID          PID  PPID  C STIME TTY          TIME CMD
root           1     0  0 Feb22 ?        00:00:43 /sbin/init splash
root           2     0  0 Feb22 ?        00:00:00 [kthreadd]
root           4     2  0 Feb22 ?        00:00:00 [kworker/0:0H]
root           6     2  0 Feb22 ?        00:00:00 [mm_percpu_wq]
root           7     2  0 Feb22 ?        00:00:00 [ksoftirqd/0]
root           8     2  0 Feb22 ?        00:00:12 [rcu_sched]
root           9     2  0 Feb22 ?        00:00:00 [rcu_bh]
root          10     2  0 Feb22 ?        00:00:00 [migration/0]
root          11     2  0 Feb22 ?        00:00:00 [watchdog/0]
root          12     2  0 Feb22 ?        00:00:00 [cpuhp/0]
root          13     2  0 Feb22 ?        00:00:00 [cpuhp/1]
root          14     2  0 Feb22 ?        00:00:00 [watchdog/1]
```

- When a user runs a process, it runs with that user's privileges, *i.e.* they can access any resource that user has permissions for
- By default, a child process inherits its parent's privileges
- Processes are isolated in memory

Process user IDs

Every process has:

- Real user ID (uid) - the user ID that started that process
- Effective user ID (euid) - the user ID that determines the process' privileges
- Saved user ID (suid) - the effective user ID before the last modification (It is possible to change the euid)

Users can change a process' IDs:

```
setuid(x)
  uid ← x
  euid ← x
  suid ← x
```

```
seteuid(x)
  uid ← uid
  euid ← x
  suid ← suid
```

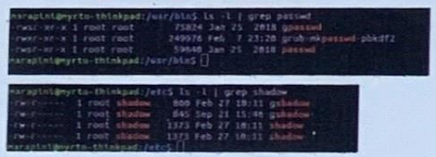
Only the euid changes!

- Root can change euid/uid to arbitrary values x;
- Unprivileged users can only change euid to uid or suid;

only drop the privileges

Elevating privileges - setuid programs

- An executable file can have the set-user-ID property (setuid) enabled
- If A executes a setuid file owned by B, then the euid of the process is B and not A
- Writing secure setuid programs is tricky because vulnerabilities may be exploited by malicious user actions
- Some programs that access system resources are owned by root and have the setuid bit set (setuid programs)



If setuid bit is set, tht. prog. runs w/ the euid of its owner, rather than the process tht. executed it.

Processes are created by a mechanism called forking, where a new process is created (that is, forked) by an existing process.

Dropping privileges with setuid

Imagine a program that runs as root and wants to fork a process with lower privileges using the following code:

```
if (auth(uid, pwd) == SUCCESS) {
  if (fork() == 0) {
    seteuid(uid);
    exec("/bin/bash");
  }
}
```

This is possible bc. after seteuid(uid), the suid does not change! Hence, unprivileged user can change euid to suid

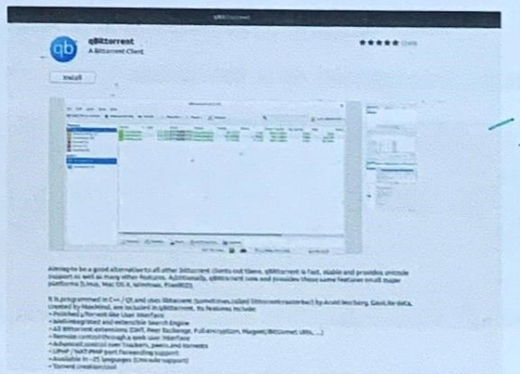
← the user can call seteuid(0) and become root!!

What we can do instead is:
setuid(uid);

UNIX permissions are too coarse-grained

Mobile OS (i.e. Android) does things differently

All application installed by a single user account have the same privileges!



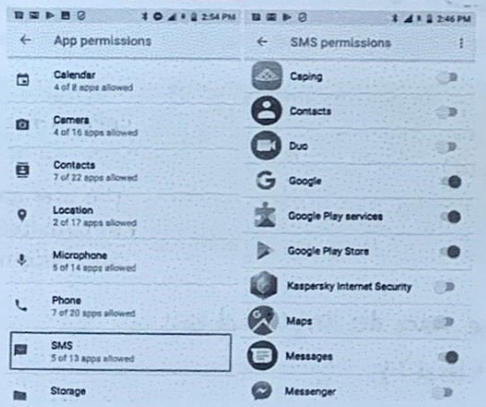
I have given to this program all my privileges

!!? What if qBittorent is malware ?!!

- Better delegate capabilities associated with specific root powers

Android permissions

- Each app runs with a different user ID
- Apps do not interact
- Permissions are set per app



As long as the OS is not compromised (the attacker does NOT have root privileges), then installing a malicious app should NOT affect other apps.

Take aways

The UNIX security model provides a simple and flexible model, but permissions are too coarse-grained:

- same permissions for all applications ran under a single user account
- many utilities have the `setuid` bit enabled → These setuid progs. are opportunities for an attacker to gain another user's privileges (i.e. root privileges)

→ many opportunities for privilege escalation attacks

→ better use capabilities when delegating privileges