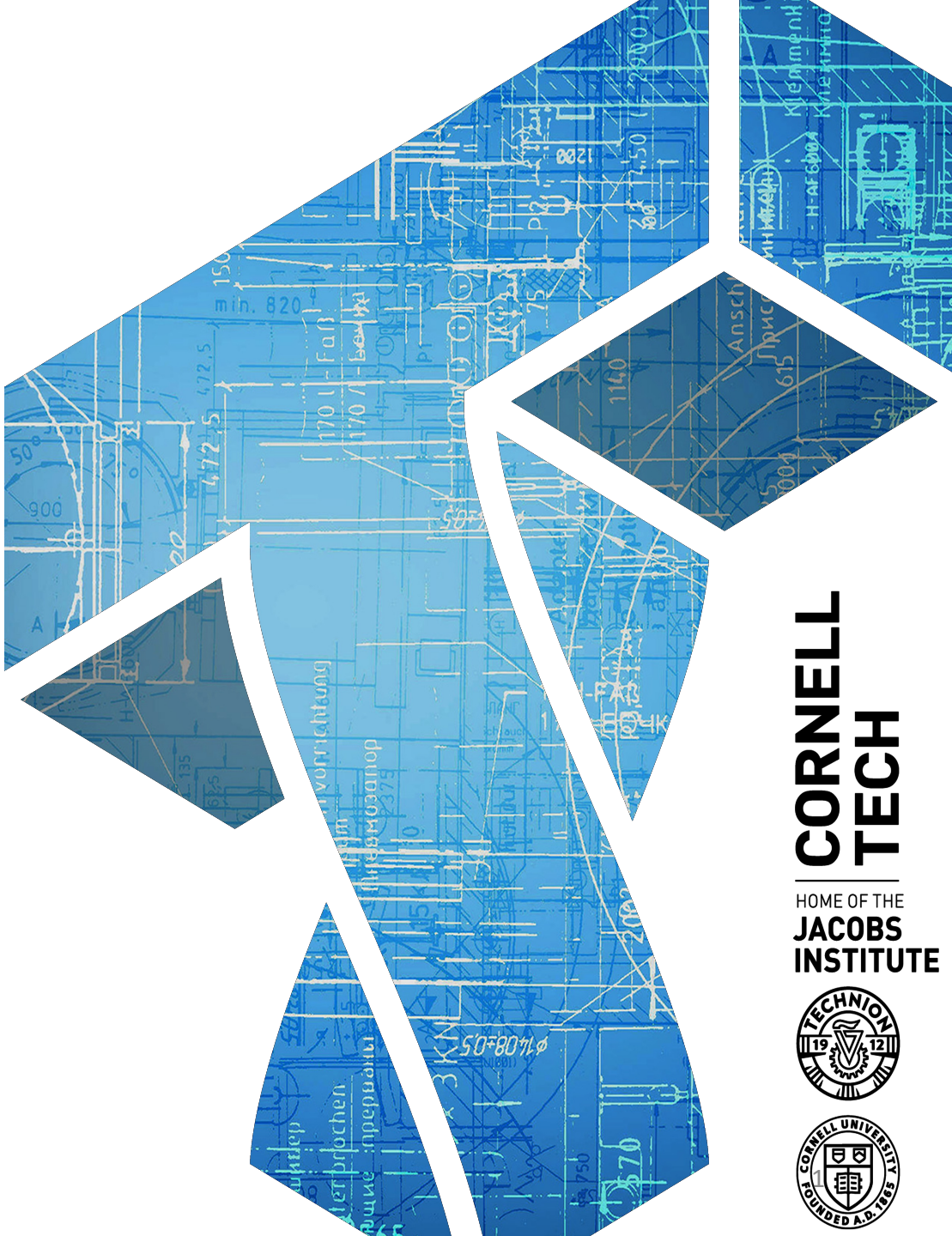
CS 5435:
OS security

Instructor: Tom Ristenpart

https://github.com/tomrist/cs5435-spring2024



CORNELL TECH

HOME OF THE JACOBS INSTITUTE

Where we're at via our web service example

Network security and crypto:
- Off-path attackers (IP hijacking, DNS poisoning, TCP injection)
- On-path attackers (compromised routers)
- Encrypted & authenticated channels (TLS)

Network
(the internet)

Web backend
(Python scripts)

Support utilities
(Shell scripts, programs)

Network or local

Database
(e.g., SQL)

Virtual machine (VM)

Client device with web browser or app

Client side adversaries:
- Account compromise
- Abuse
- Web vulnerabilities

OS and low-level software security:
- UNIX style access controls
- OS privilege levels
- Buffer overflows and memory corruption

Take yourself back to the 1960's…

Time-share multi-user computers coming into use

GE-645
36 bit address space
Up to 4 processors
Magnetic tape drives
Supported virtual memory in hardware

Courtesy of
https://www.bell-labs.com/var/articles/invention-unix/

Multiplexed Information and Computing Service (Multics)

Project to develop operating system for time-shared systems

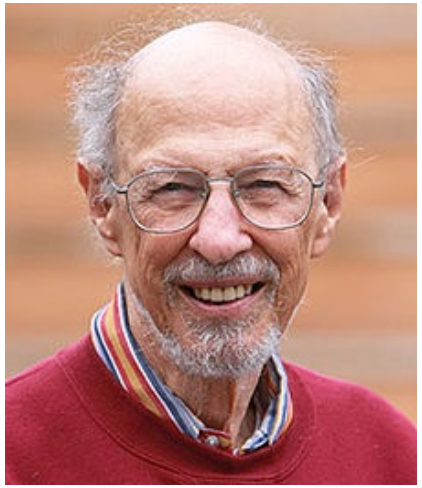
- Designed from 1964-1967.
- MIT project MAC, Bell Labs, and GE
- ~100 installations at greatest extent
- Last one shut down in 2000 (Canadian department of defense)

> "A small but useful hardware complement would be 2 CPU units, 128K of core, 4 million words of high speed drum, 16 million words of disc, 8 tapes, 2 card readers, 2 line printers, 1 card punch and 30 consoles."
> [Vyssotsky, Corbato, Graham 1965]

Multics: ancestor to many OS's

Lots of innovations in design

- Use of segmentation and virtual memory with hardware support
- SMP (shared memory multiprocessor)
- Written in PL/1 (high level language)
- Inspired developers of Unix

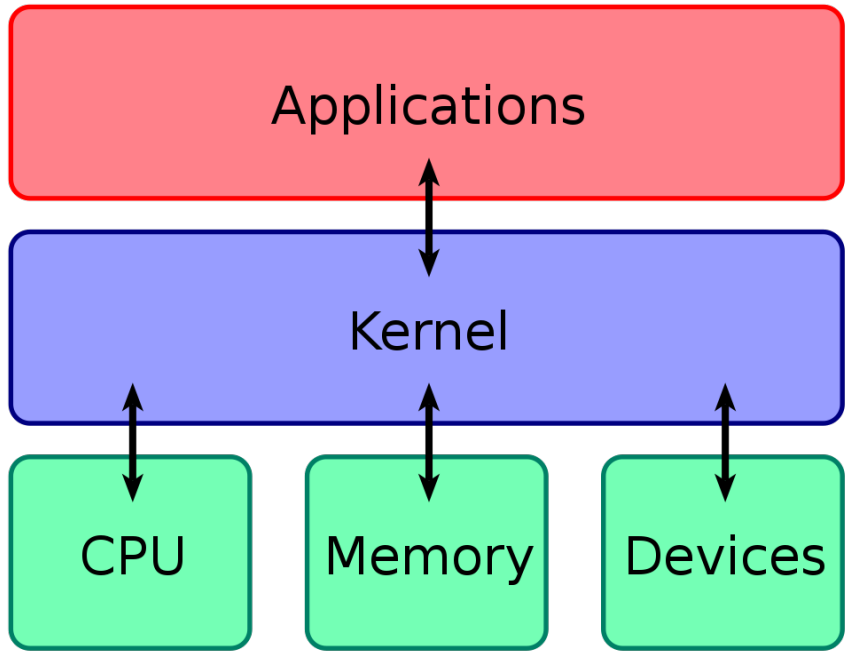


F. Corbato

Significant attention paid to security

Recall operating system basics

- Multi-tasking, multi-user OS are now the norm
- ***Kernel*** mediates between applications and resources
- ***Applications*** consist of one or more processes
- ***Processes*** have executable program, allocated memory, resource descriptors (e.g., file descriptors), processor state

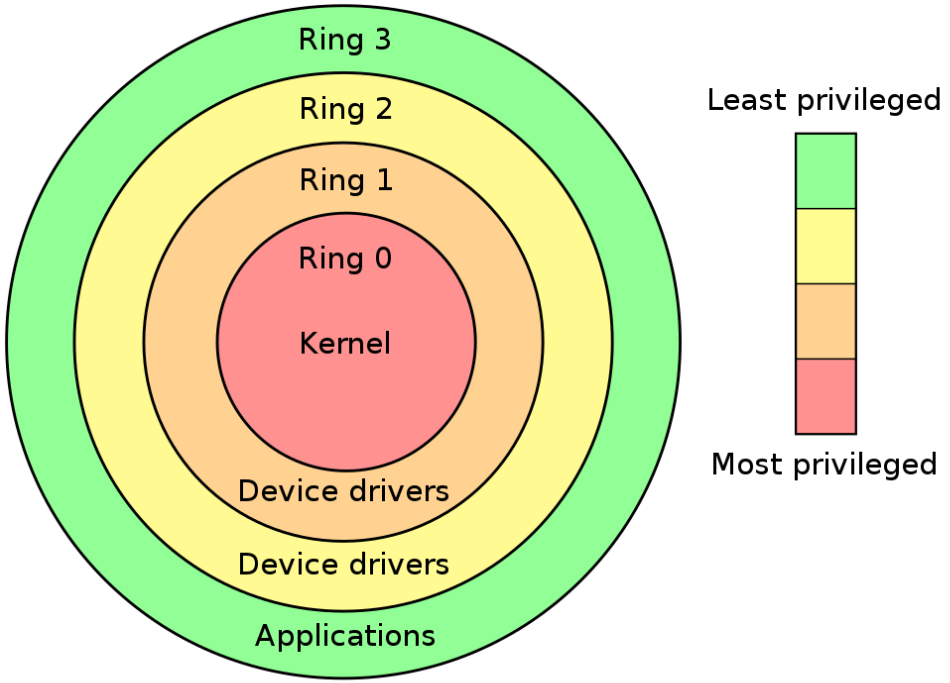


Privilege levels and protection rings

Different parts of system must operate at different privilege levels

Protection rings included in all typical CPUs today and used by most operating systems

- Lower number = higher privilege
- Ring 0 is supervisor
- Inherit privileges over higher levels

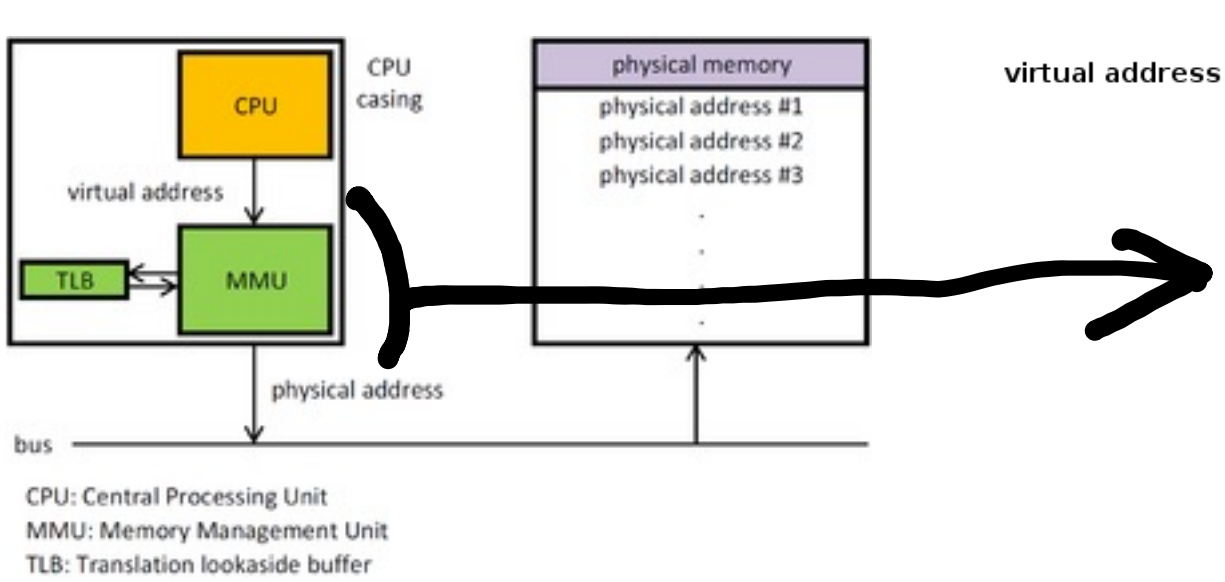


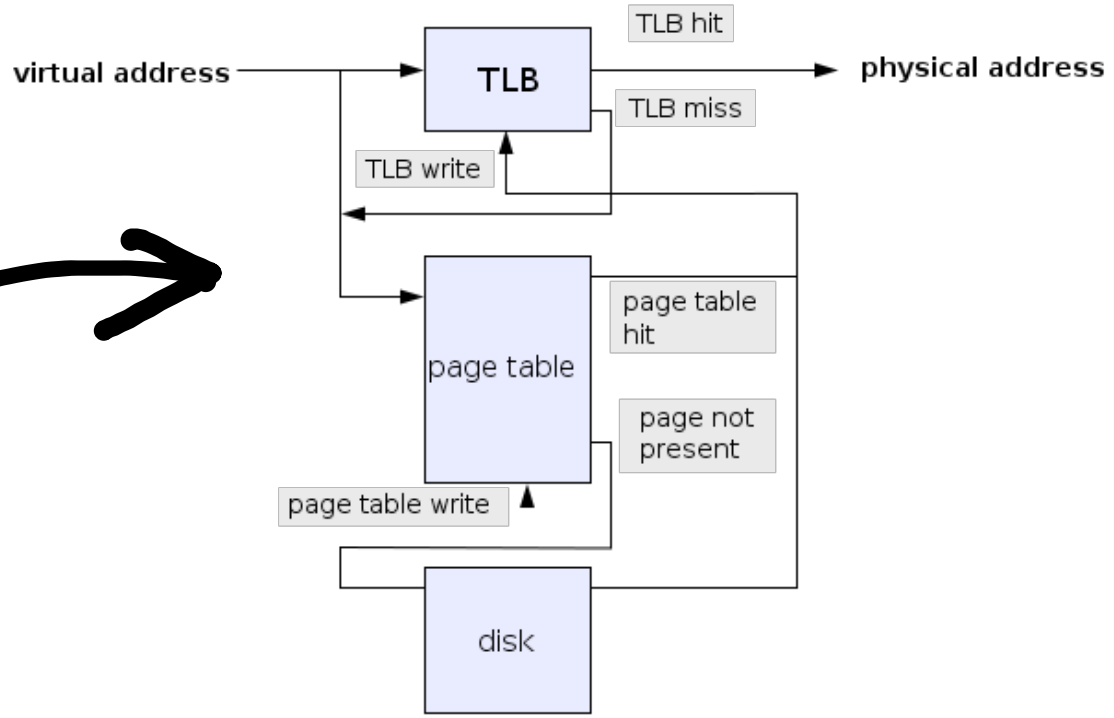
Intel x86 protection rings

Principle of least privilege:

User account or process should have least privilege level required to perform their intended functions

Memory Management





What prevents a process from reading another process's memory?

Systems security ingredients

- Security model:
  - Abstraction of system to help us express security policies
- Security policies:
  - Specification of what parts of system should be allowed to do
- Security mechanism:
  - How we implement the policy

Systems security ingredients

- Security model:
  - Subjects
  - Objects
  - Operations

	Unix	Web
Subjects (Who)	Users, Processes	Domains
Objects (What)	Memory, Files, Hardware devices …	DOM components, cookies, …
Operations	Read, write, execute	Read, write, …

Access control matrix

Most frequent way of specifying security policy

Objects

Subjects	**file 1**	**file 2**	**…**	**file n**
user 1	read, write	read, write, own		read
user 2				
…				
user m	append	read, execute		read,write, own

User i has permissions for file j as indicated in cell [i,j]

Due originally to Lampson in 1971

Two common implementation paradigms

	file 1	file 2	…	file n
user 1	read, write	read, write, own		read
user 2				
…				
user m	append	read, execute		read,write, own

(1) Access control lists

Column stored with file

(2) Capabilities

Row stored for each user

Unforgeable tickets given to user

Discretionary access controls: users can set some access controls (e.g., Unix file system ACLs)

Mandatory access controls: access controls set in one centralized location (e.g., SElinux)

UNIX Users

Service accounts used to run background processes (e.g., web server)

User accounts

- Typically tied to a specific human
- Every user has a unique integer ID (UID)

Many system operations can only run as root

Users and Superusers

A user has username, group name, password

Root is an administrator / superuser (UID 0)

- Can read and write any file or system resource (network, etc.)
- Can modify the operating system
- Can become any other user
  - Execute commands under any other user’s ID
- Can the superuser read passwords?

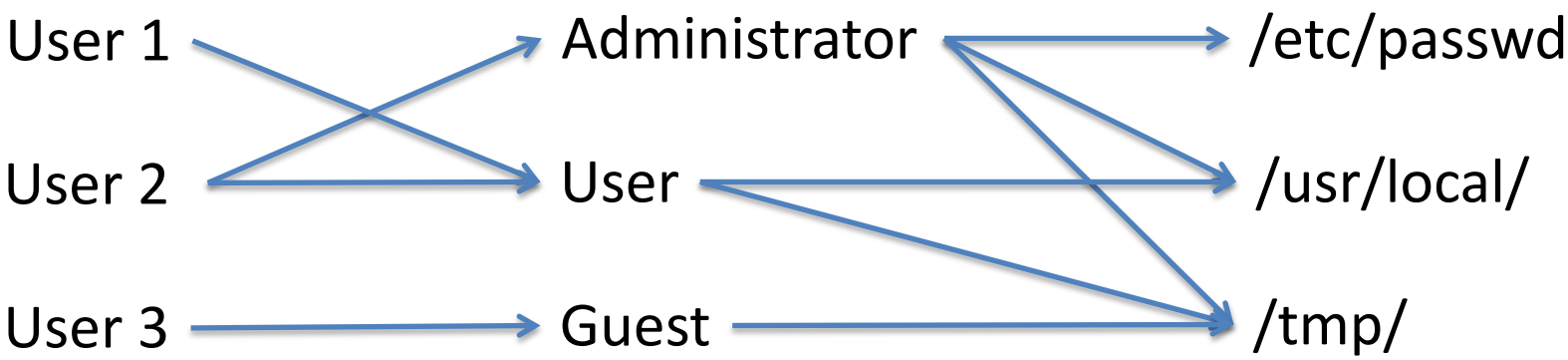
Roles (groups)

Group is a set of users

Administrator User Guest

Simplifies assignment of permissions at scale

Concept sometimes referred to as role-based access control (RBAC)



UNIX file permissions

- Owner, group
- Permissions set by owner or root
- Resolving permissions:
  - If user=owner, then owner privileges
  - If user in group, then group privileges
  - Otherwise, all privileges

Access control in UNIX

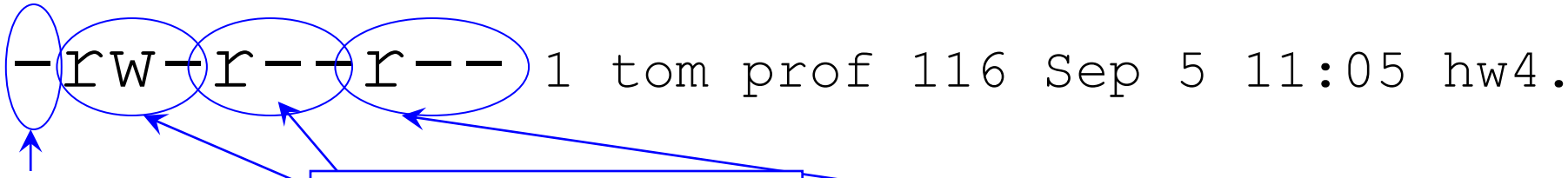
Everything is a file

- Files and also sockets, pipes, hardware devices….
- Files are laid out in a tree

inode data structure records OS management information about the file

- UID and GID of the file owner
- Type, size, location on disk
- Time of last access (atime), last inode modification (ctime), last file contents modification (mtime)
- Discretionary ACL via permission bits

UNIX Permission Bits



File type
- regular file
d directory
b block file
c character file
l symbolic link
p pipe
s socket

Permission bits
r read
w write
x execute (if directory, traverse it)
s setuid, setgid (if directory, files have gid of dir owner)
t sticky bit (if directory, append-only)

Each file has 12 ACL bits:
rwx for each of owner, group, all; set user ID; set group ID; sticky bit

UNIX Process Permissions

Process (normally) runs with the permissions of the user who invoked process

Suppose user wants to change password…

- Need to modify /etc/shadow password file
- /etc/shadow is owned by root
- Can user's process modify /etc/shadow?
- How does passwd program change user's password?

Process IDs in UNIX

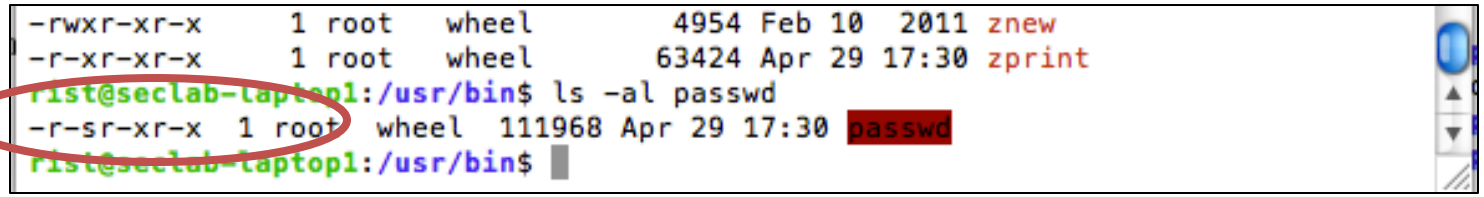
Each process has three UIDs (similar for GIDs)

- **Real ID:** user who started the process
- **Effective ID:** determines effective access rights of the process
- **Saved ID:** used to swap IDs, gaining or losing privileges

Known as setuid programs

If an executable's setuid bit is set, it will run with the effective privileges of its owner, not the user who started it

- Example: when I run passwd, real UID is tom (13630), effective UID is root (0), saved UID is tom (13630)



Acquiring and Dropping Privilege

- To acquire privilege, assign privileged UID to effective ID
- To drop privilege temporarily, remove privileged UID from effective ID and store it in saved ID
  - Can restore it later from saved ID
- To drop privilege permanently, remove privileged UID from both effective and saved ID

Example:

- Apache Web Server must start as **root** because only root can create a socket that listens on port 80 (a privileged port)
- Without privilege reduction, any Apache bug would give attacker root access to server
- Instead, Apache creates children like this:

```
if (fork() == 0) {
    int sock = socket(":80");
    setuid(getuid("www-data"));
}
```

Setuid management is tricky and error-prone!

seteuid system call

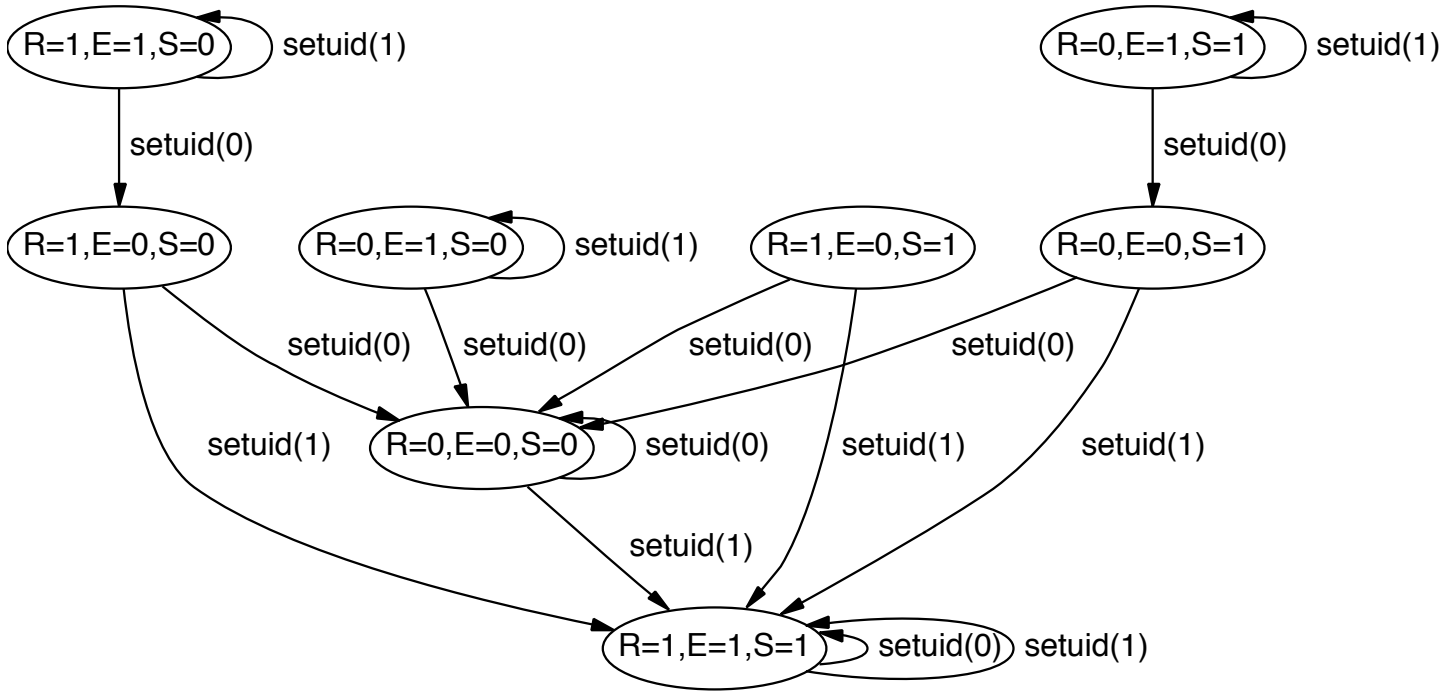
```
uid = getuid();
eid = geteuid();
seteuid(uid);       // Drop privileges
…
seteuid(eid);       // Raise privileges
file = fopen( “/etc/shadow”, “w” );
…
seteuid(uid);       // drop privileges
```

seteuid can:
- go to SUID or RUID always
- any ID if EUID is 0

Note: this is C pseudocode
C still used widely for OS’s and low-level programming

Details of setuid more complicated

Chen, Wagner, Dean “Setuid Demystified”



(a) An FSA describing *setuid* in Linux 2.4.18

Setuid allows necessarily privilege escalation but…

- Source of many ***privilege escalation vulnerabilities***
  - Bug that allows lower-privilege user to perform actions as higher-privilege user (most often: root)

Race conditions

Control-flow hijacking vulnerabilities in local setuid program gives privilege escalation

Checking Access Rights

User should only be able to access a file if they have permission to do so

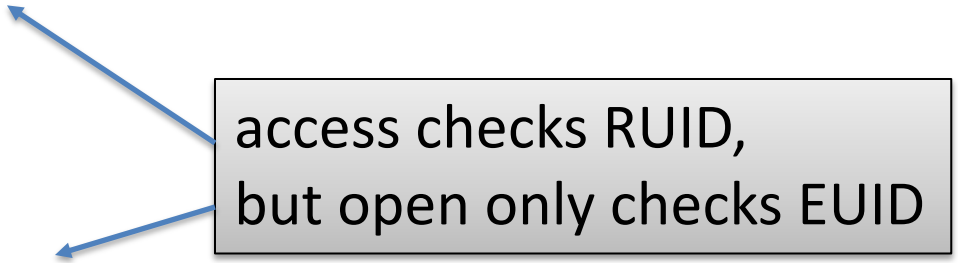
But what if the user is running as setuid-root?

- For example, the printing program usually runs with root privileges so it can access the printer… but root can read any file! How does the printing program know that the user who invoked it has the right to read (and print) a given file?

UNIX has a special access() system call

Race conditions
Time-of-check-to-time-of-use (TOCTTOU)

```
if( access(“/tmp/myfile”, R_OK) != 0 ) {
    exit(-1);
}
file = open( “/tmp/myfile”, “r” );
read( file, buf, 100 );
close( file );
print( “%s\n”, buf );
```



Say program is setuid root:

access checks RUID, but open only checks EUID

Time

```
if( access("/tmp/myfile", R_OK) != 0 ) {
        exit(-1);
}
file = open( "/tmp/myfile", "r" );
read( file, buf, 100 );
close( file );
print( "%s\n", buf );
```

```
access("/tmp/myfile", R_OK)
```

```
ln –s /etc/shadow /tmp/myfile
```

```
open( "/tmp/myfile", "r" );
```

```
print( "%s\n", buf );
```

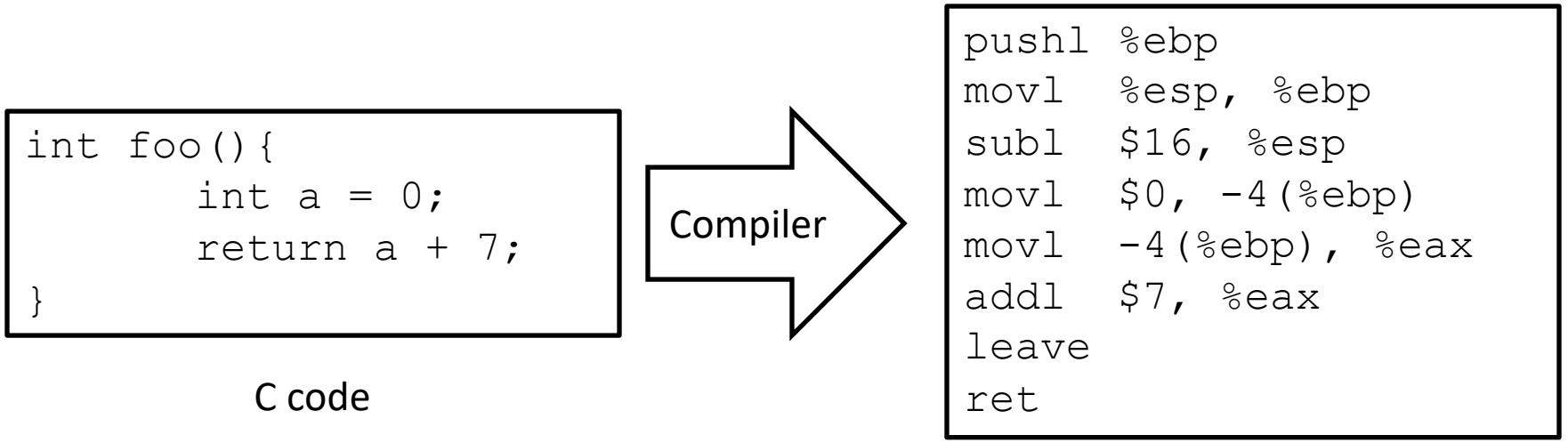
Example of concurrency vulnerability

Prints out shadow file (including password hashes)

Better code

```
euid = geteuid();
ruid = getuid();
seteuid(ruid);              // drop privileges
file = open( “/tmp/myfile”, “r” );
read( file, buf, 100 );
close( file );
print( “%s\n”, buf );
```

Low-level software vulnerabilities



Control flow refers to sequence of instructions followed by CPU

Control flow hijacking:

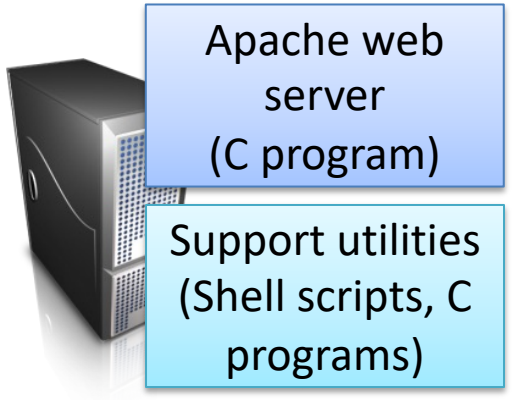
- exploiting vulnerability to adversarially manipulate control flow

Low-level software vulnerabilities

C programs notorious for vulnerabilities that allow ***control flow hijacking***

Local privilege escalation:

- Given user-level access, obtain root-level access
- Typical exploit gives attacker root shell



Web server

Remote exploit:

- Gain illicit access remotely over the network
- Typical exploit gives attacker remote shell

Summary

- Multics: seminal multi-user operating system
  - many security features
  - significant auditing performed, achieved high security certifications
- Security models, policies, mechanisms
  - Subjects, Objects, Operations
  - Access control lists
- Unix security
  - File system permissions
  - Setuid programs
- Race conditions, intro to control-flow hijacking