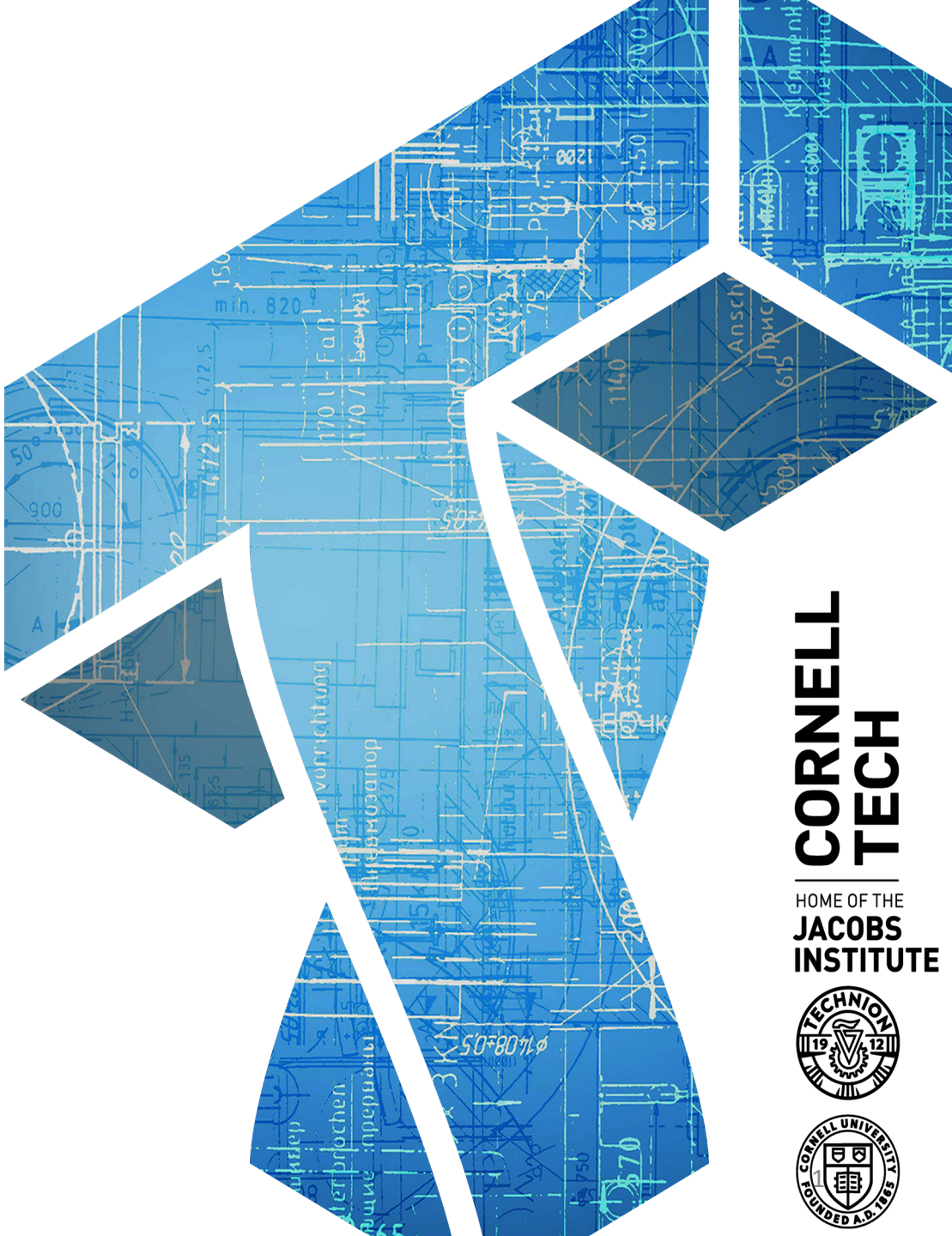
CS 5435:
OS security

Instructor: Tom Ristenpart

https://github.com/tomrist/cs5435-fall2019



CORNELL TECH

HOME OF THE JACOBS INSTITUTE

Announcements

- HW3 due date extended to Monday, Nov 11
- Office hours next couple days
- HW4 will be released later this week

- No lecture on Monday Nov 11 (holiday)
- Possibly no lecture on Wednesday Nov 13, may do review session for HW4

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)

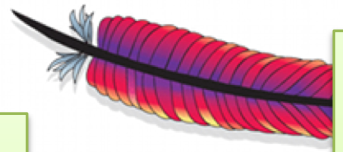
Network or local

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

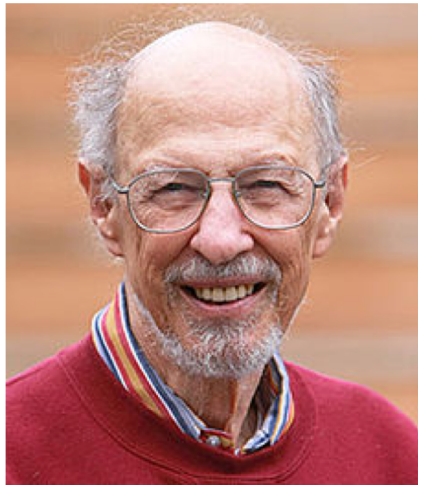
- Designed from 1964-1967.
- MIT project MAC, Bell Labs, and GE
- ~100 installations at greatest extent
- Last one shut down in 2000 (Canadian department of defense)

> "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

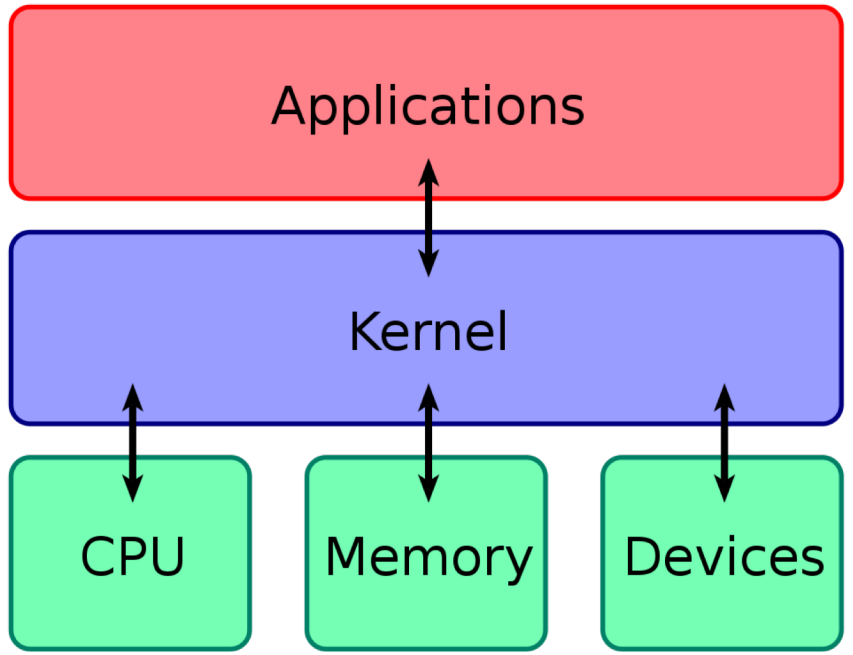


F. Corbato

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

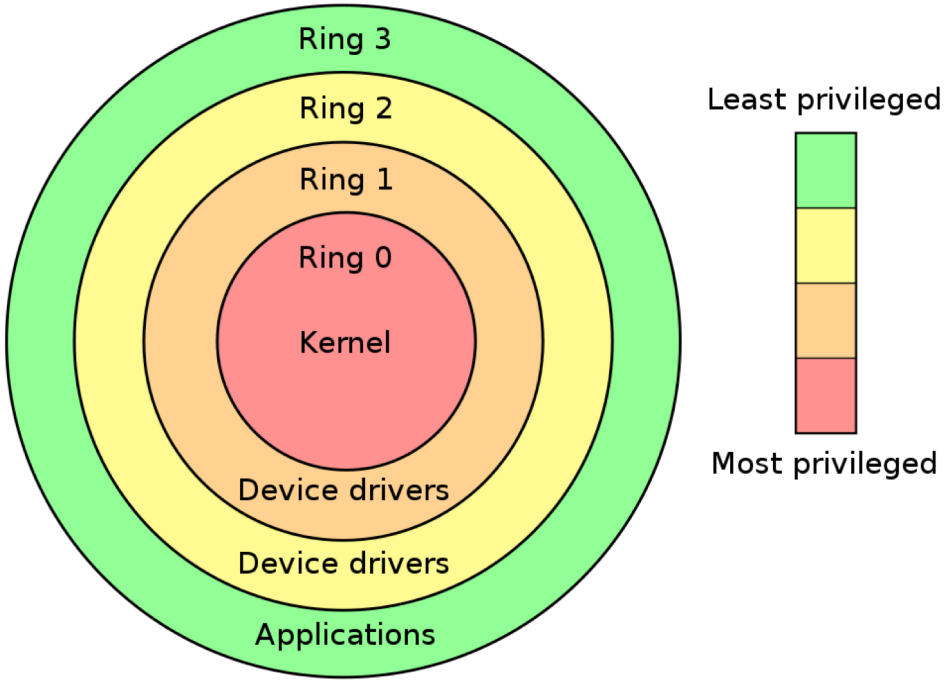


Privilege levels and protection rings

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

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 Security Model

- Users include high-privilege ***root (aka superuser)***
  - User ID (UID) is 0
- Users authenticated traditionally via passwords
  - /etc/passwd
  - /etc/shadow
    - Contains password hashes, restricted access

	Unix
Subjects (Who)	Users, Processes
Objects (What)	Memory, Files, Hardware devices …
Operations	Read, write, execute

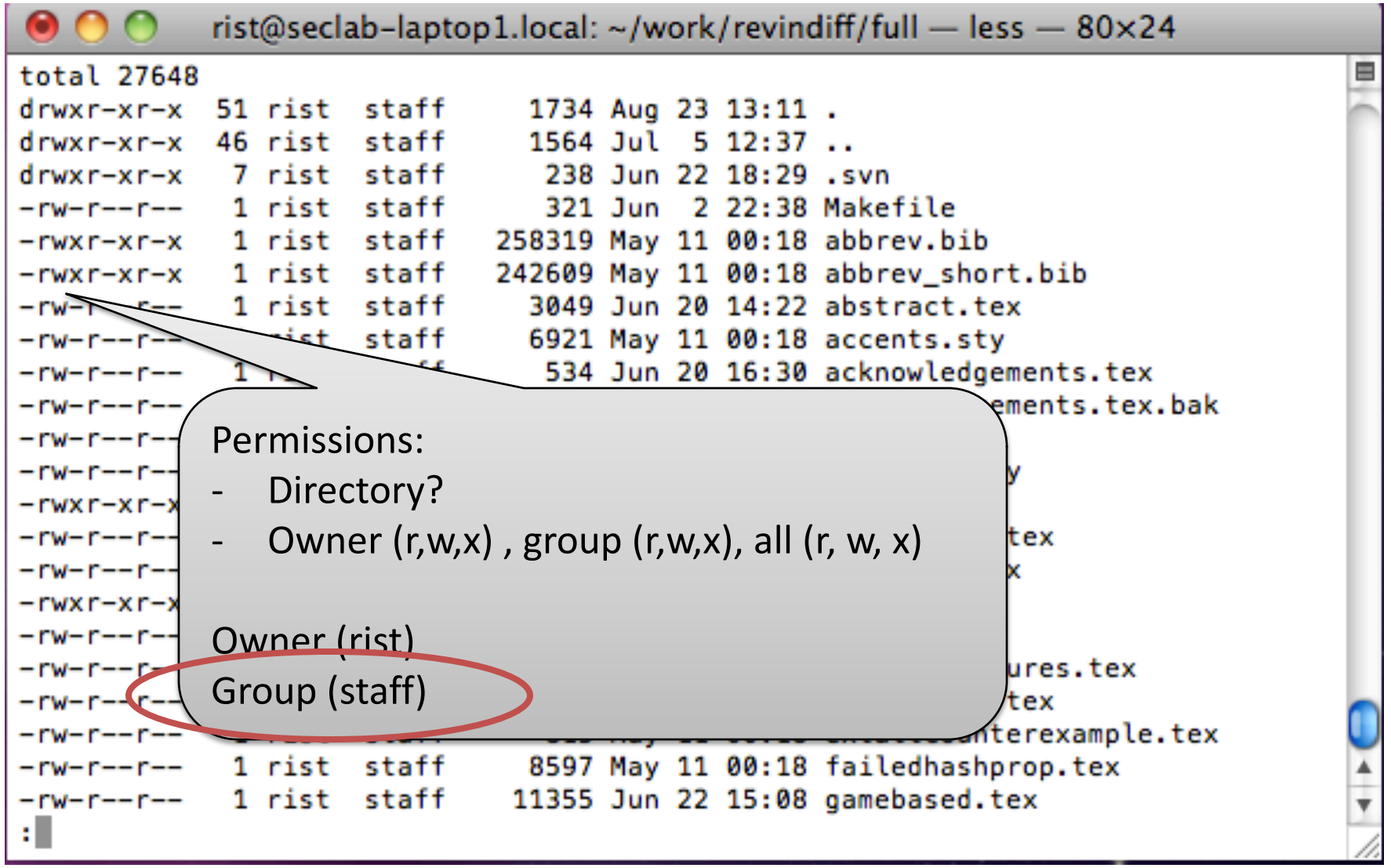
Unix Security Model

- File system permissions uses discretionary ACL approach
- Each file has 12 ACL bits
  - rwx for each of owner, group, all
  - Set user ID
  - Set group ID
  - Sticky bit

	Unix
Subjects (Who)	Users, Processes
Objects (What)	Memory, Files, Hardware devices …
Operations	Read, write, execute

- 0 means user with write on directory can rename/remove file
- 1 means only file owner, directory owner, root can do so

UNIX-style file system ACLs



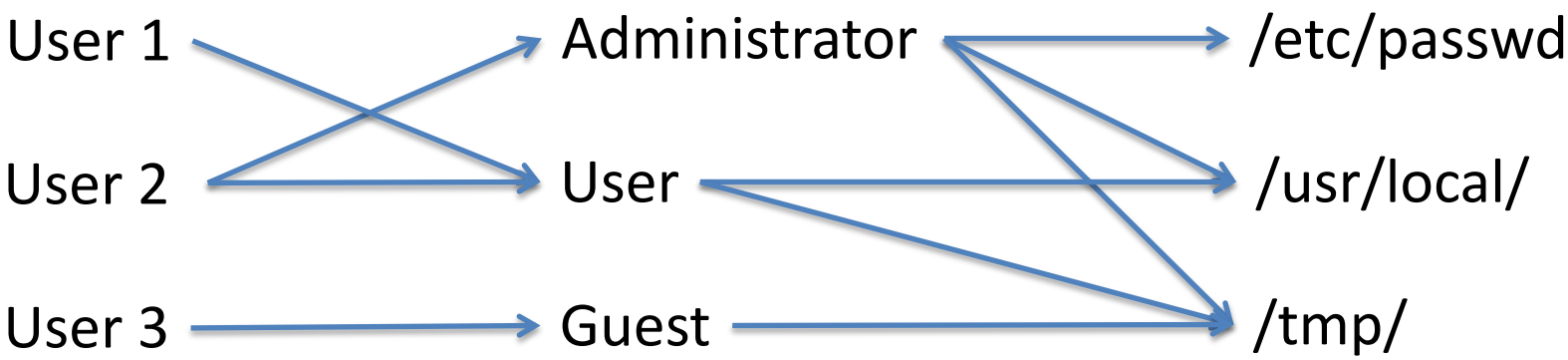
Roles (groups)

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)

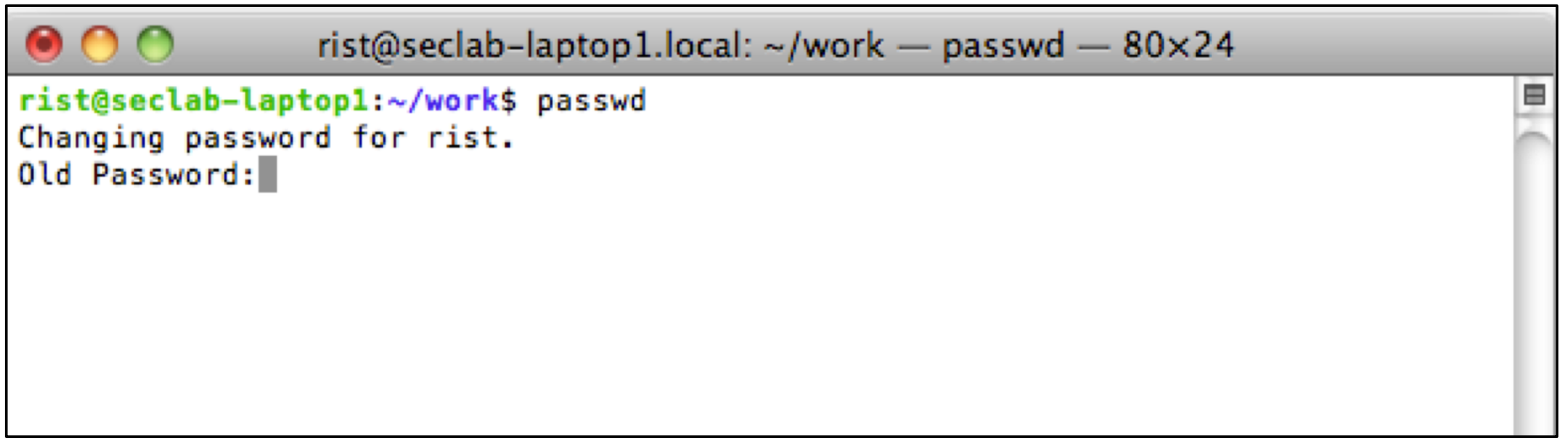


UNIX file permissions

- 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

UNIX Process permissions

- Process (normally) runs with permissions of user that invoked process

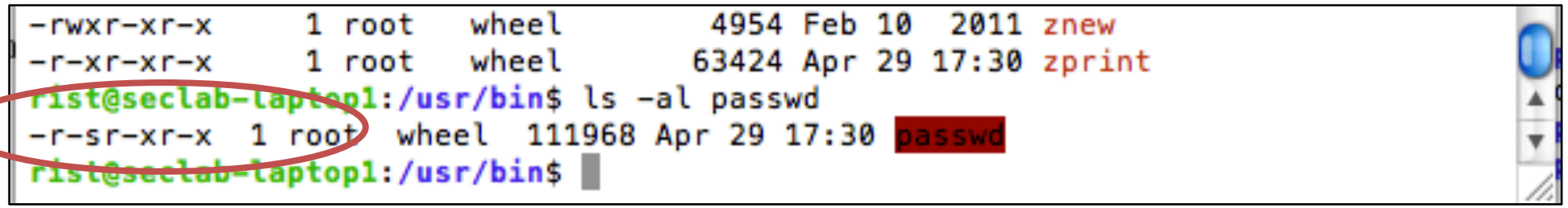


/etc/shadow is owned by root

Users shouldn’t be able to write to it generally

How can passwd reset user’s password?

Setuid programs



- setuid bit – execute with privileges of file's owner
- setgid bit – execute with privileges of file's group

- So passwd is a ***setuid program*** program runs at permission level of owner, not user that runs it

Least privilege at *process granularity*:
passwd runs as root to access /etc/shadow
Can we do better?

Process permissions

Process (normally) runs with permissions of user that invoked process
Unless executable is setuid or setguid

Real user ID (RUID) --
same as UID of parent (who started process)
Used to figure out who started process

passwd example:
RUID = rist

Also SGID, EGID, etc..

Effective user ID (EUID) --
from set user ID bit of file being executed or due to sys call
indicates current permissions of process

passwd example:
EUID = root

Saved user ID (SUID) --
place to save the previous UID if one temporarily changes it

seteuid system call

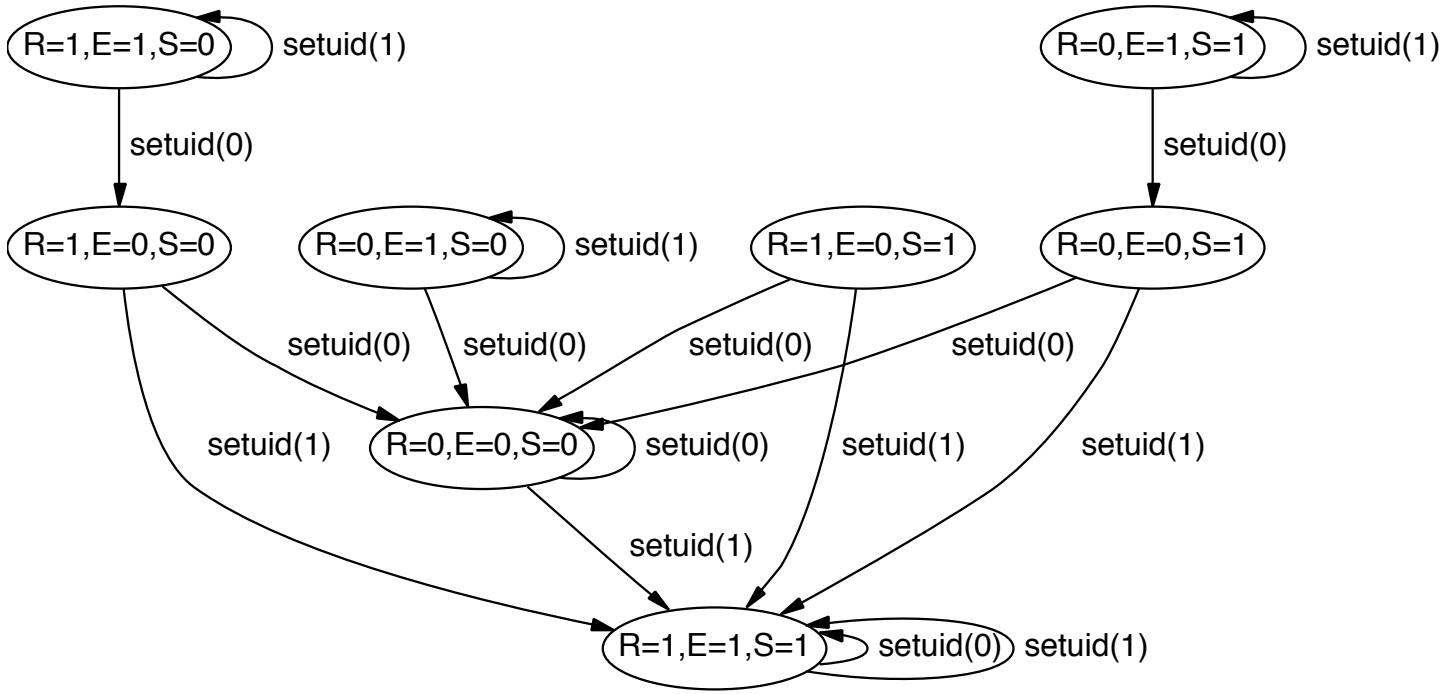
```
uid = getuid();
eid = geteuid();
seteuid(uid);       // Drop privileges
…
seteuid(eid);       // Raise privileges
file = fopen( “/etc/shadow”, “w” );
…
seteuid(uid);       // drop privileges
```

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…

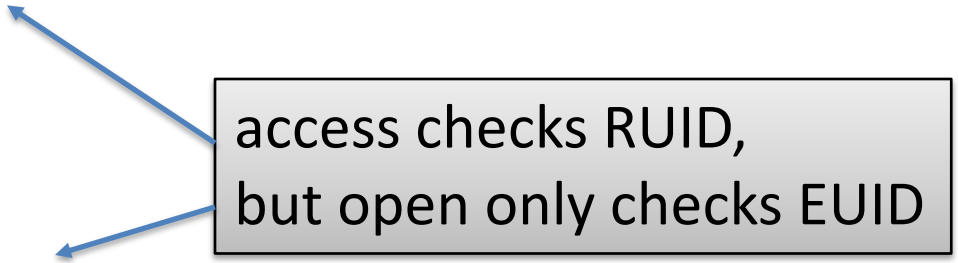
- Source of many ***privilege escalation vulnerabilities***
  - Bug that allows lower-privilege user to perform actions as higher-privilege user (most often: root)

Race conditions

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

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

access("/tmp/myfile", R_OK)

ln –s /etc/shadow /tmp/myfile

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

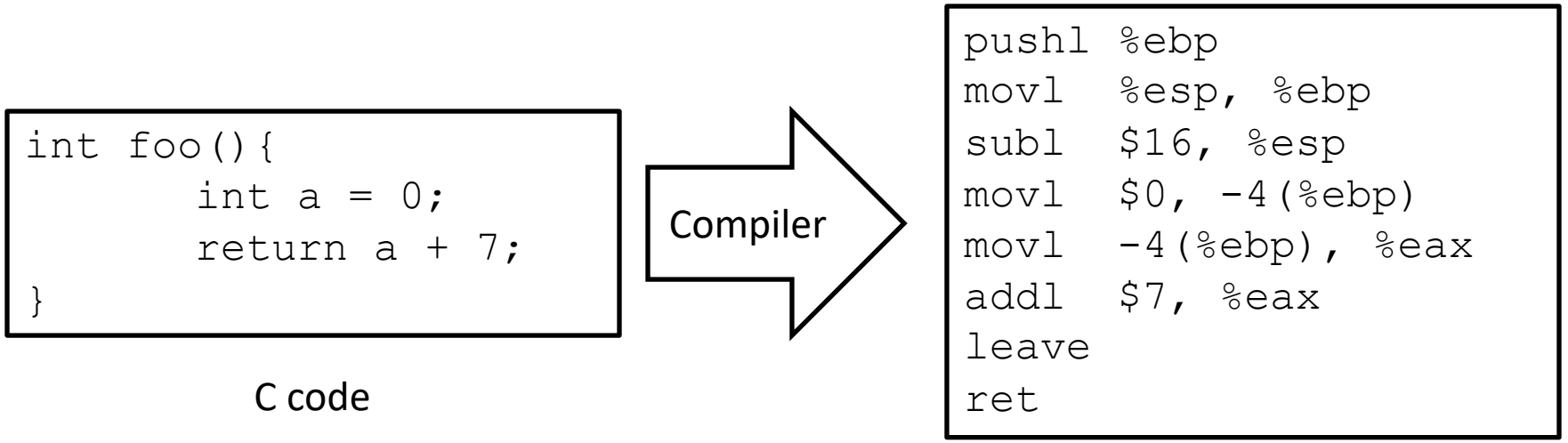
print("%s\n", buf);

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



C code

Disassembled machine code

Control flow refers to sequence of instructions followed by CPU

Control flow hijacking:

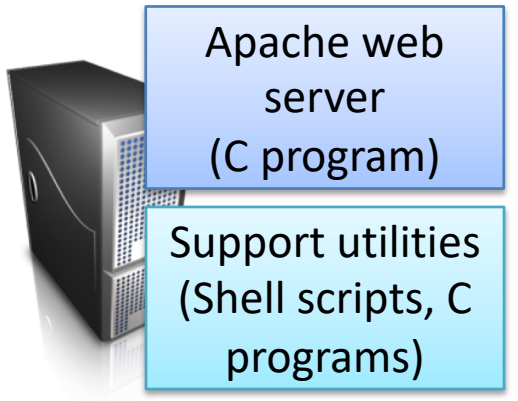
- exploiting vulnerability to adversarially manipulate control flow

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