

# Information Security

## CS 526



### Topic 7: Operating Systems Security Basics & Unix Access Control

# Readings for This Lecture

- Wikipedia
  - [CPU modes](#)
  - [System call](#)
  - [Filesystem Permissions](#)
- Other readings
  - UNIX File and Directory Permissions and Modes
    - <http://www.hccfl.edu/pollock/AUnix1/FilePermissions.htm>
  - Unix file permissions
    - <http://www.unix.com/tips-tutorials/19060-unix-file-permissions.html>

# Announcements and Outline

- Outline of this topic
  - Brief overview of the goals of OS security
  - Memory protection, CPU modes, and system calls
  - Access control basic concepts
  - Security principles
  - UNIX File System permissions
  - UNIX processes

# What Security Goals Does Operating System Provide?

- Originally: time-sharing computers: enabling multiple users to securely share a computer
  - Separation and sharing of processes, memory, files, devices, etc.
- What is the threat model?
  - Users may be malicious, users have terminal access to computers, software may be malicious/buggy, and so on
- Security mechanisms
  - Memory protection
  - Processor modes
  - User authentication
  - File access control

# What Security Goals Does Operating System Provide?

- More recent past and present: Networked desktop computers: ensure secure operation in networked environment
- New threat?
  - Attackers coming from the network. Network-facing programs on computers may be buggy. Users may be hurt via online communication.
- Security mechanisms
  - Authentication; Access Control
  - Secure Communication (using cryptography)
  - Logging & Auditing
  - Intrusion Prevention and Detection
  - Recovery

# What Security Goals Does Operating System Provide?

- Present and near future: mobile computing devices:
- New threat?
  - Apps (programs) may be malicious or questionable.
  - More tightly connected with personal life of the owner.
- Security mechanisms?
  - Isolation of each app.
  - Help users assess risks of apps.
  - Risk communication.

# Memory Protection: Access Control to Memory

- Ensures that one user's process cannot access other's memory
  - fence
  - relocation
  - base/bounds register
  - segmentation
  - paging
  - ...
- Operating system and user processes need to have different privileges

# CPU Modes (a.k.a. processor modes or privilege)

- System mode (privileged mode, master mode, supervisor mode, kernel mode)
  - Can execute any instruction
  - Can access any memory locations, e.g., accessing hardware devices,
  - Can enable and disable interrupts,
  - Can change privileged processor state,
  - Can access memory management units,
  - Can modify registers for various descriptor tables .

Reading: [http://en.wikipedia.org/wiki/CPU\\_modes](http://en.wikipedia.org/wiki/CPU_modes)



# User Mode

- User mode
  - Access to memory is limited,
  - Cannot execute some instructions
  - Cannot disable interrupts,
  - Cannot change arbitrary processor state,
  - Cannot access memory management units
- Transition from user mode to system mode can only happen via well defined entry points, i.e., through system calls

Reading: [http://en.wikipedia.org/wiki/CPU\\_modes](http://en.wikipedia.org/wiki/CPU_modes)

# System Calls

- Guarded gates from user mode (space, land) into kernel mode (space, land)
  - use a special CPU instruction (often an interruption), transfers control to predefined entry point in more privileged code; allows the more privileged code to specify where it will be entered as well as important processor state at the time of entry.
  - the higher privileged code, by examining processor state set by the less privileged code and/or its stack, determines what is being requested and whether to allow it.

[http://en.wikipedia.org/wiki/System\\_call](http://en.wikipedia.org/wiki/System_call)

# Kernel space vs User space

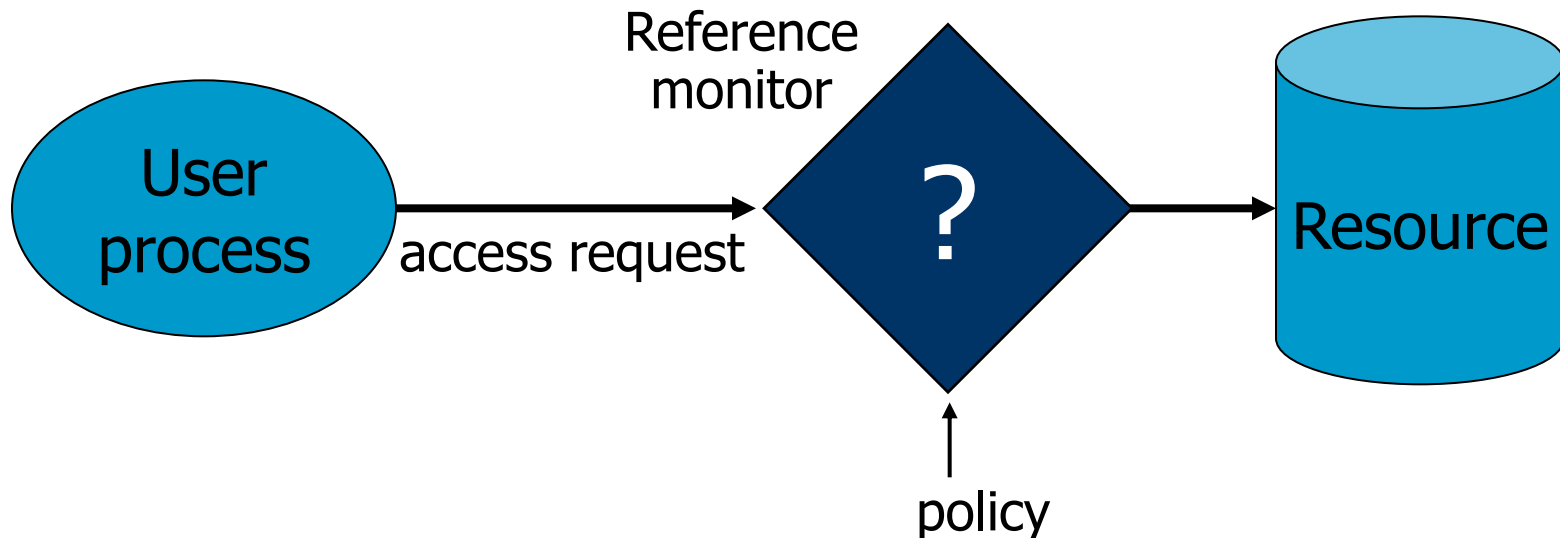
- Part of the OS runs in the kernel model
  - known as the **OS kernel**
- Other parts of the OS run in the user mode, including service programs (daemon programs), user applications, etc.
  - they run as **processes**
  - they form the user space (or the user land)
- **Why root user is more powerful than normal users?**

# Privilege Levels

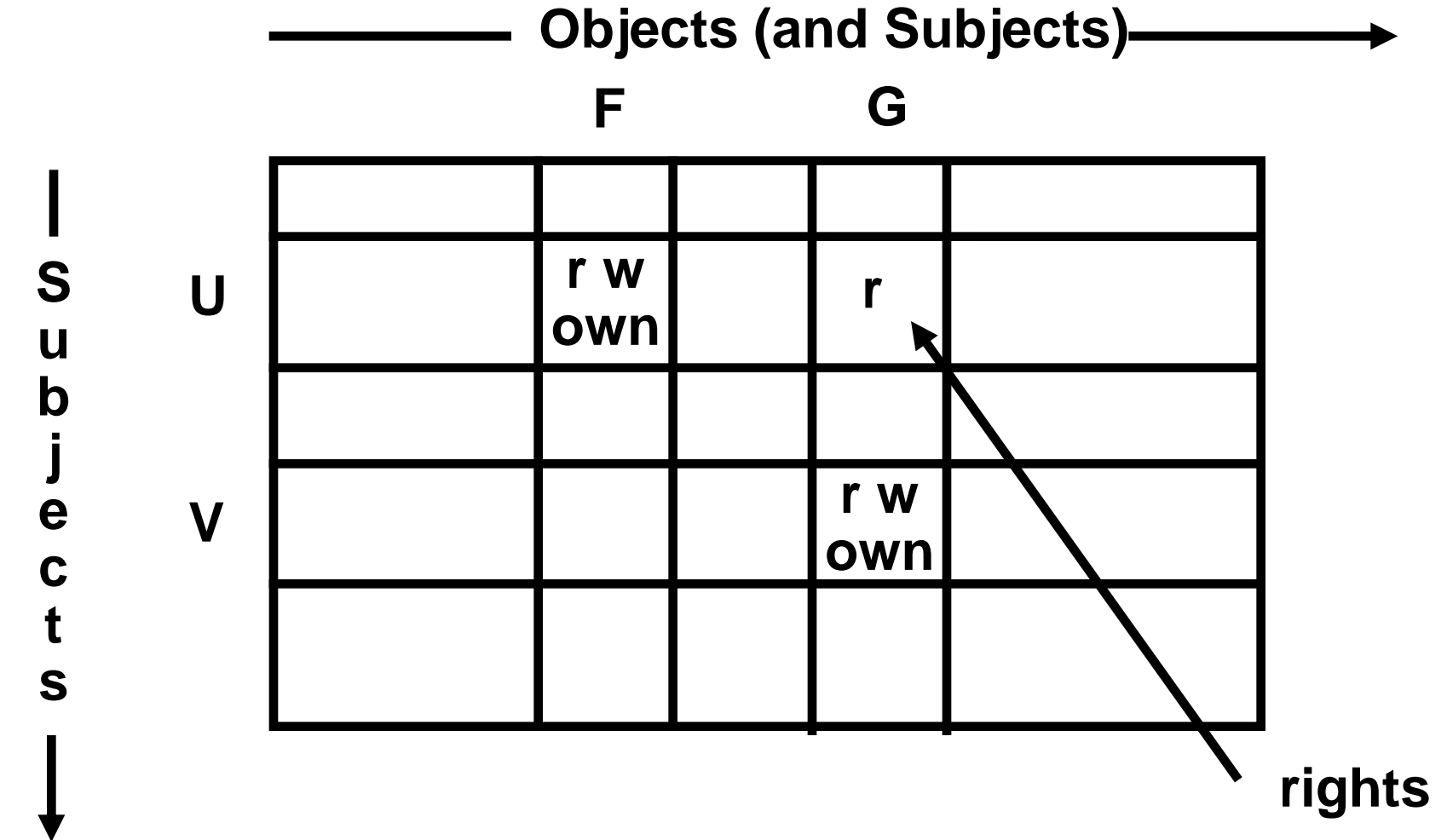
- Security is often achieved by running control/protection code at a higher privilege level
- Components running at the same level can be isolated by a higher-privilege component
- If attack and defense are at the same level, then it is an arms' race and there can be no guarantee

# Access control

- A **reference monitor** mediates all access to resources
  - Principle: Complete mediation: control **all** accesses to resources



# ACCESS MATRIX MODEL



# ACCESS MATRIX MODEL

- Basic Abstractions
  - Subjects
  - Objects
  - Rights
- The rights in a cell specify the access of the subject (row) to the object (column)

# PRINCIPALS AND SUBJECTS

- A subject is a program (application) executing on behalf of some principal(s)
- A principal may at any time be idle, or have one or more subjects executing on its behalf

**What are subjects in UNIX?**

**What are principals in UNIX?**



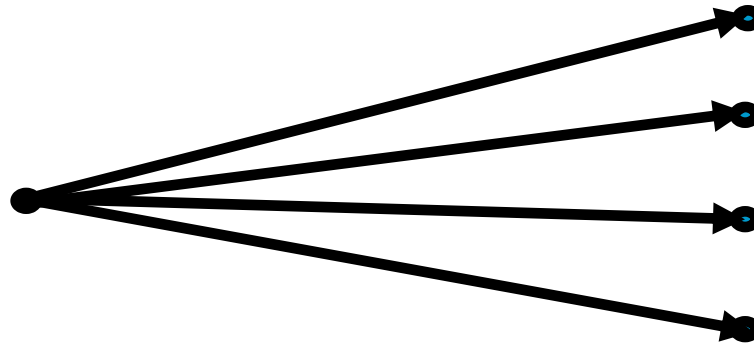
# OBJECTS

- An object is anything on which a subject can perform operations (mediated by rights)
- Usually objects are passive, for example:
  - File
  - Directory (or Folder)
  - Memory segment
- But, subjects (i.e. processes) can also be objects, with operations performed on them
  - kill, suspend, resume, send interprocess communication, etc.

# Basic Concepts of UNIX Access Control: Users, Groups, Files, Processes

- Each user account has a unique UID
  - The UID 0 means the super user (system admin)
- A user account belongs to multiple groups
- Subjects are processes
  - associated with uid/gid pairs, e.g., (euid, egid), (ruid, rgid), (suid, sgid)
- Objects are files

# USERS AND PRINCIPALS



**USERS**

**PRINCIPALS**

**Real World User**

**Unit of Access Control  
and Authorization**

the system authenticates the human user to  
a particular principal

# USERS AND PRINCIPALS

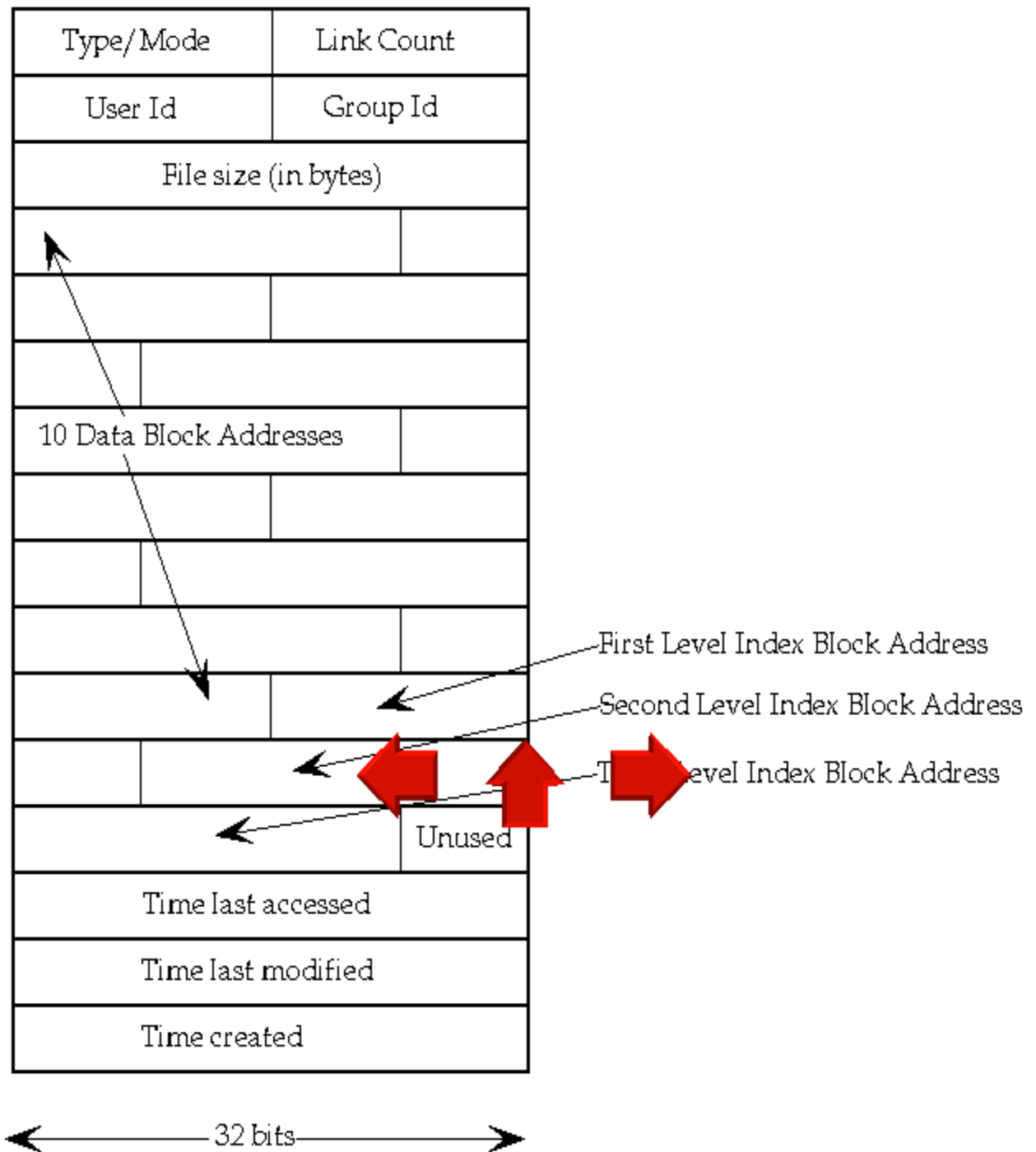
- There should be a one-to-many mapping from users to principals
  - a user may have many principals, but
  - each principal is associated with an unique user
- This ensures accountability of a user's actions

**What does the above imply in UNIX?**

# Organization of Objects

- In UNIX, almost all objects are modeled as files
  - Files are arranged in a hierarchy
  - Files exist in directories
  - Directories are also one kind of files
- Each object has
  - owner
  - group
  - 12 permission bits
    - rwx for owner, rwx for group, and rwx for others
    - suid, sgid, sticky

UNIX  
inodes:  
Each file  
corresponds  
to an inode

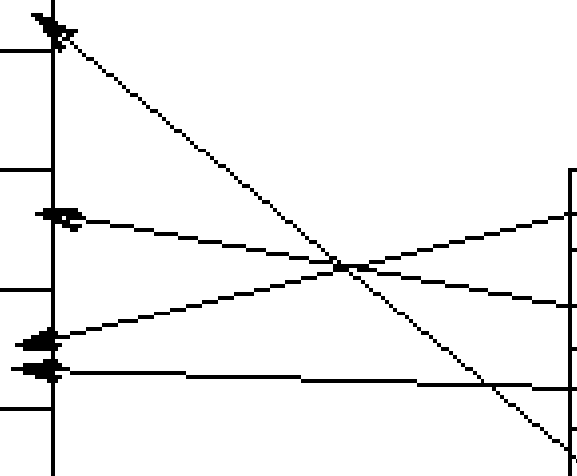


# UNIX Directories

Inode table


Directory

i1	name1
i2	name2
i1	name3
i4	name4



# Basic Permissions Bits on Files (Non-directories)

- Read controls reading the content of a file
  - i.e., the read system call
- Write controls changing the content of a file
  - i.e., the write system call
- Execute controls loading the file in memory and execute
  - i.e., the execve system call



# Permission Bits on Directories

- Read bit allows one to show file names in a directory
- The execution bit controls traversing a directory
  - does a lookup, allows one to find inode # from file name
  - `chdir` to a directory requires execution
- Write + execution control creating/deleting files in the directory
  - Deleting a file under a directory requires no permission on the file
- Accessing a file identified by a path name requires execution to all directories along the path

# The suid, sgid, sticky bits

	suid	sgid	sticky bit
non-executable files	no effect	affect locking (unimportant for us)	not used anymore
executable files	change euid when executing the file	change egid when executing the file	not used anymore
directories	no effect	new files inherit group of the directory	only the owner of a file can delete

# Some Examples

- What permissions are needed to access a file/directory?
  - read a file: /d1/d2/f3
  - write a file: /d1/d2/f3
  - delete a file: /d1/d2/f3
  - rename a file: from /d1/d2/f3 to /d1/d2/f4
  - ...
- File/Directory Access Control is by System Calls
  - e.g., open(2), stat(2), read(2), write(2), chmod(2), opendir(2), readdir(2), readlink(2), chdir(2), ...

# The Three Sets of Permission Bits

- Intuition:
  - if the user is the owner of a file, then the r/w/x bits for owner apply
  - otherwise, if the user belongs to the group the file belongs to, then the r/w/x bits for group apply
  - otherwise, the r/w/x bits for others apply

# Other Issues On Objects in UNIX

- Accesses other than read/write/execute
  - Who can change the permission bits?
    - The owner can
  - Who can change the owner?
    - Only the superuser
- Rights not related to a file
  - Affecting another process
  - Operations such as shutting down the system, mounting a new file system, listening on a low port
    - traditionally reserved for the root user

# Security Principles Related to Access Control

- Psychological acceptability is related to configuring access control policies.
- Complete mediation
- Fail-safe defaults
- Least privilege

# Principle of Complete mediation

- Every access to every object must be checked for authority. This principle, when systematically applied, is the primary underpinning of the protection system. It forces a system-wide view of access control, which in addition to normal operation includes initialization, recovery, shutdown, and maintenance. It implies that a foolproof method of identifying the source of every request must be devised. It also requires that proposals to gain performance by remembering the result of an authority check be examined skeptically. If a change in authority occurs, such remembered results must be systematically updated.

# How Complete Mediation Can Be Violated

- Any system that can directly access disk may bypass file-system level access control



# Principle of Fail-safe defaults

- Base access decisions on permission rather than exclusion. This principle[...] means that the default situation is lack of access, and the protection scheme identifies conditions under which access is permitted. The alternative, in which mechanisms attempt to identify conditions under which access should be refused, presents the wrong psychological base for secure system design. A conservative design must be based on arguments why objects should be accessible, rather than why they should not.

# Application of Fail-Safe Defaults

- E.g., whitelisting instead of black listing, e.g., for anti-virus products
  - Challenge, deal with novel unknown
- Firewall policies use default deny
- A guest account with privilege violates fail-safe defaults

# Principle of Least Privilege

- Every program and every user of the system should operate using the least set of privileges necessary to complete the job. Primarily, this principle limits the damage that can result from an accident or error. It also reduces the number of potential interactions among privileged programs to the minimum for correct operation, so that unintentional, unwanted, or improper uses of privilege are less likely to occur. [.....] The military security rule of "need-to-know" is an example of this principle.

# Subjects vs. Principals

- Access rights are specified for user accounts (principals).
- Accesses are performed by processes (subjects)
- The OS needs to know on which user accounts' behalf a process is executing
- How is this done in Unix?

# Process User ID Model in Modern UNIX Systems

- Each process has three user IDs
  - real user ID (ruid)                      owner of the process
  - effective user ID (euid)                  used in most access control decisions
  - saved user ID (suid)
- and three group IDs
  - real group ID
  - effective group ID
  - saved group ID

# Process User ID Model in Modern UNIX Systems

- When a process is created by *fork*
  - it inherits all three users IDs from its parent process
- When a process executes a file by *exec*
  - it keeps its three user IDs unless the set-user-ID bit of the file is set, in which case the effective uid and saved uid are assigned the user ID of the owner of the file
- In addition, a process may change the user ids via system calls

# The Need for suid/sgid Bits

- Some operations are not modeled as files and require user id = 0
  - halting the system
  - bind/listen on “privileged ports” (TCP/UDP ports below 1024)
  - non-root users need these privileges
- File level access control is not fine-grained enough
- System integrity requires more than controlling who can write, but also how it is written

# Security Problems of Programs with suid/sgid

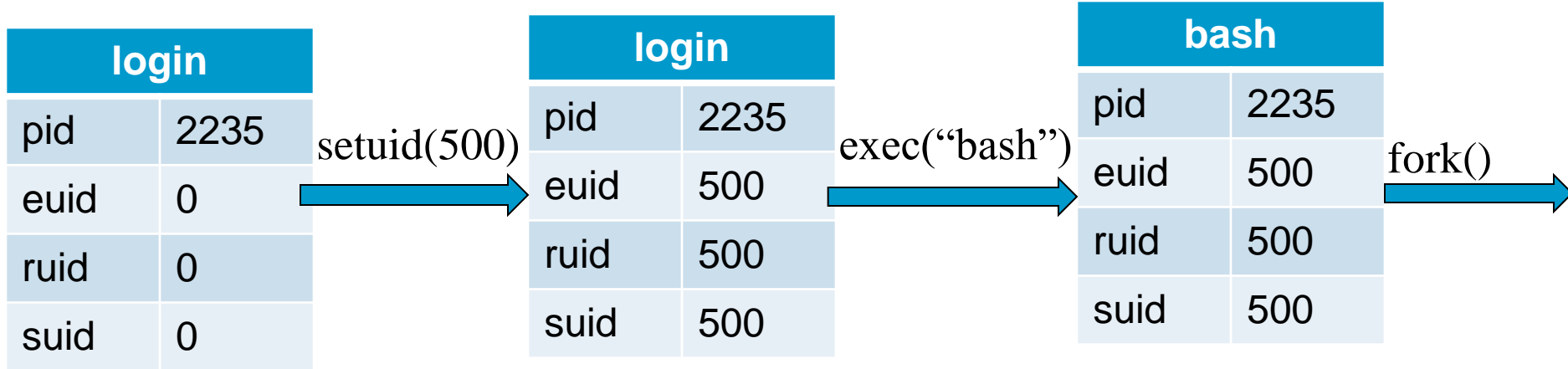
- These programs are typically setuid root
- Violates the least privilege principle
  - every program and every user should operate using the least privilege necessary to complete the job
- Why violating least privilege is bad?
- How would an attacker exploit this problem?
- How to solve this problem?



# Changing effective user IDs

- A process that executes a set-uid program can drop its privilege; it can
  - drop privilege permanently
    - removes the privileged user id from all three user IDs
  - drop privilege temporarily
    - removes the privileged user ID from its effective uid but stores it in its saved uid, later the process may restore privilege by restoring privileged user ID in its effective uid

# What Happens during Logging in



After the login process verifies that the entered password is correct, it issues a `setuid(500)` system call.

The login process then loads the shell, giving the user a login shell.

The user types in the `passwd` command to change his password.

bash	
pid	2235
eid	500
ruid	500
suid	500

bash	
pid	2297
eid	500
ruid	500
suid	500

exec("passwd")



passwd	
pid	2297
eid	0
ruid	500
suid	0

Drop privilege permanently

passwd	
pid	2297
eid	500
ruid	500
suid	500

Drop privilege temporarily

passwd	
pid	2297
eid	500
ruid	500
suid	0

The fork call creates a new process, which loads "passwd", which is owned by root user, and has setuid bit set.

# Coming Attractions ...

- Software Vulnerabilities

