


**PURDUE UNIVERSITY**

# Computer Security

## CS 426

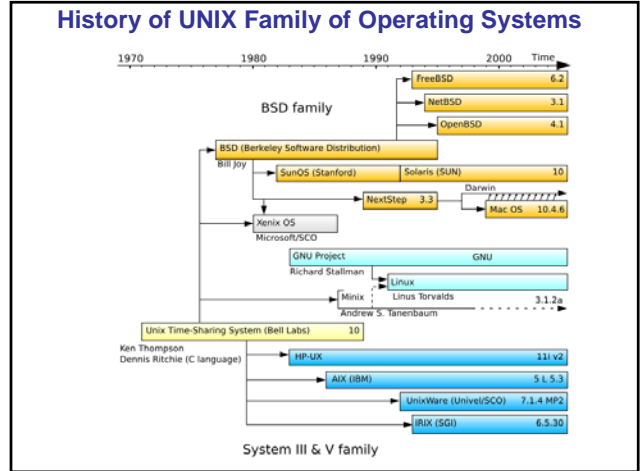
### Lecture 14

## Access Control in Unix Systems



**Elisa Bertino**  
Purdue University  
IN, USA  
bertino@cs.purdue.edu

1



**PURDUE UNIVERSITY**

## Users, Groups, Processes, Files

- Each user has a unique UID
- Each group has a unique GID
- Each process has a unique PID
- Users belong to multiple groups GID
- Objects whose access is controlled
  - Files
  - Directories

3

**PURDUE UNIVERSITY**

## Organization of Objects

- Files are arranged in a hierarchy
- Files exist in directories
- Directories are one type of file
- In UNIX, accesses on directories are not inherited

4

## Basic Permissions Bits on Files

- 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
- Many operations can be performed only by the owner of the file

5

## Where are Permission Bits Kept?

- Each file/directory has associated an inode.
- The file type, permissions, owner UID and owner GID are saved on disk in the inode of a file or directory

6

## 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/creating 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

7

## Sticky Bit

- For files its use is obsolete
- For a directory:
  - it prevents users from renaming, moving or deleting contained files owned by users other than themselves, even if they have write permission to the directory
  - only the directory owner and superuser can rename, move, or delete files within a directory with the sticky bit set

8

## Permission Bits: Example

- drwxr-xr-x
- First: directory or not
- Next three: owner permission
- Next three: group permission
- Next three: others permission

9

## The Three Sets of Permission Bits

- Order:
  - 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

10

## Users vs. Subjects

- The above discussions are about what users (accounts) can access a file
- Actions are performed by subjects (processes)
- When a subject accesses a file, the system needs to know which user it is acting on behalf of

11

## Example

- Consider the **passwd** program
- Needs to update a system-wide password file, which ordinary users should not be able to modify, but only root can modify
- Needs to be run by ordinary users

How to support such cases?

12

## Real User ID vs. Effective User ID

- Super-user ID is 0 and has no restrictions, any user with ID 0 would have the super-user rights
- Each process has three user IDs
  - real user ID (ruid): owner of the process
  - effective user ID (euid): used in most access control decisions, often the same as ruid unless there is a change
  - saved user ID (suid): keeps the previous euid if it was a change
- and three group IDs
  - real group ID
  - effective group ID
  - saved group ID

13

## setuid

- **setuid:**
  - allows a program running on behalf of one user operate as if it were running as another user (for example root)
  - allows certain processes to have more than ordinary privileges while still being executable by ordinary users
- **How?** The effective user identifier takes the value of the owner of the file

14

## 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*
  - if (set-user-ID bit is not set)
    - it keeps its three user IDs
  - otherwise // set-user-ID bit of the file is set
    - euid = ruid
    - suid = previous euid
- How to fix address in the passwd example?
- Passwd is owned by root and setuid is set
- When a user executes it, then the effective user becomes the root, so the program runs as root on behalf of the user

15

## An Example of a Security Problem of Programs with suid/sgid

- If a user leaves your terminal unattended, an unscrupulous passerby can destroy the security of the user account simply by typing the commands:
 

```
% cp /bin/sh /tmp/break-acct
% chmod 4755 /tmp/break-acct
```
- These commands create a SUID version of the sh program. Whenever the attacker runs this program, the attacker becomes the user - with full access to all of user files and privileges.
- The attacker might even copy this SUID program into a hidden directory so that it would only be found if the superuser scanned the entire disk for SUID programs.
- Not all system administrators do such scanning on any regular basis.

In the above example:  
4 specifies *set user ID* and the rest is equivalent to u=rwx (4+2+1),go=rx (4+1 & 4+1).<sup>16</sup>

## Summary of UNIX Access Control

- Users, subjects, objects
  - how objects are organized
  - how to associate subjects with users
- Using the suid/sgid bits relies on the trustworthiness of programs
- Programs may have bugs
- One attempt to fix it is by providing the mechanism for temporary dropping privileges