CSci 4271W Development of Secure Software Systems Day 10: Unix Access Control

Stephen McCamant University of Minnesota, Computer Science & Engineering

Outline

Access control: mechanism and policy Unix filesystem concepts Announcements intermission Unix permissions basics Exercise: using Unix permissions More Unix permissions

Configurability

- Basic idea: let one mechanism (implementation) support a variety of security policies
- I.e., make security a system configuration
- Classic example for today: OS access control
- Flexible mechanism to support different policies
- Trade-off: an incorrect configuration can lead to insecurity

Confidentiality and integrity

- Access control directly serves two security goals:
- Confidentiality, opposite of information disclosure
- Integrity, opposite of tampering
- By prohibiting read and write operations respectively

Access control policy

- Decision-making aspect of OS
- Should subject S (user or process) be allowed to access object (e.g., file) O?
- Complex, since administrator must specify what should happen

Access control matrix

	grades.txt	/dev/hda	/usr/bin/bcvi
Alice	r	rw	rx
Bob	rw	-	rx
Carol	r	-	rx

Slicing the matrix O(nm) matrix impractical to store, much less administer Columns: access control list (ACL) Convenient to store with object E.g., Unix file permissions Rows: capabilities Convenient to store by subject E.g., Unix file descriptors

Groups/roles

- Simplify by factoring out commonality
- Before: users have permissions
- After: users have roles, roles have permissions
- Simple example: Unix groups
- Complex versions called role-based access control (RBAC)

Outline

Access control: mechanism and policy

Unix filesystem concepts

Announcements intermission

- Unix permissions basics
- Exercise: using Unix permissions
- More Unix permissions

One namespace

- All files can be accessed via absolute pathnames made of directory components separated by slashes
- I.e., everything is a descendant of a root directory named /

Filesystems and mounting

- There may be multiple filesystems, like disk partitions or removable devices
- One filesystem is the root filesystem that includes the root directory
- Other filesystems are mounted in place of a directory
 - E.g., /media/smccaman/mp3player/podcast.mp3

Special files and devices

- Some hardware devices (disks, serial ports) also look like files
 - Usually kept under /dev
- Some special data sources look like devices
 dev/null,/dev/zero,/dev/urandom
- Some OS data also available via /proc and sys filesystems
 - E.g., /proc/self/maps

Current directory, relative paths

At a given moment, each process has a current working directory

Shanged by cd shell command, chdir system call

Pathnames that do not start with / are interpreted relative to the current directory

Inodes

- Most information about a file is a structure called an inode
- Includes size, owner, permissions, and a unique inode number
- Inodes exist independently of pathnames

Directory entries and links

- A directory is a list of directory entries, each mapping from a name to an inode
- These mappings are also called links
- "Deleting a file" is really removing a directory entry
 The system call unlink

Entries . and ..

- Every directory contains entries named . and ...
- links back to the directory itself
- Inks back to the parent directory, or itself for the root

(Hard) links

Multiple directory entries can link to the same inode
 These are called hard links

- These are called hard links
- Only allowed within one filesystem, and not for directories

Symbolic links

- Symbolic links are a different linking method
- A symbolic link is an inode that contains a pathname
- Most system calls follow symbolic as well as hard links to operate on they point to

Outline

Access control: mechanism and policy

Unix filesystem concepts

Announcements intermission

Unix permissions basics

Exercise: using Unix permissions

More Unix permissions

Midterm-related resources

- Four solution set PDFs from old exams are now posted
- Bring your questions (including lab and pset-related) to office hours or Piazza

Midterm-related advice

Pencil or erasable pen would be good writing implements (unless you don't make mistakes)

You can bring any paper, but distilling the most useful information will save you time

Several previous exams had questions related to terminology: this can benefit from targeted studying

Outline

Access control: mechanism and policy Unix filesystem concepts Announcements intermission Unix permissions basics Exercise: using Unix permissions

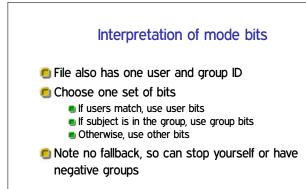
More Unix permissions

UIDs and GIDs

- To kernel, users and groups are just numeric identifiers
- Names are a user-space nicety
- E.g., /etc/passwd mapping
- Historically 16-bit, now 32
- User O is the special superuser root
 - Exempt from all access control checks

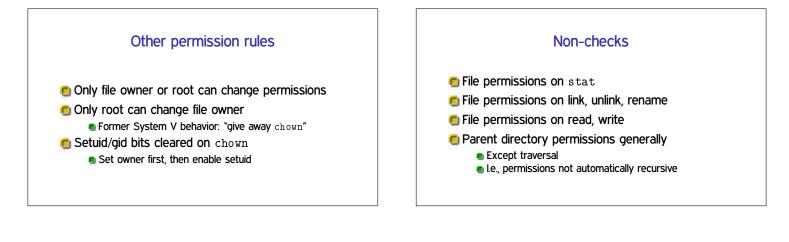
File mode bits

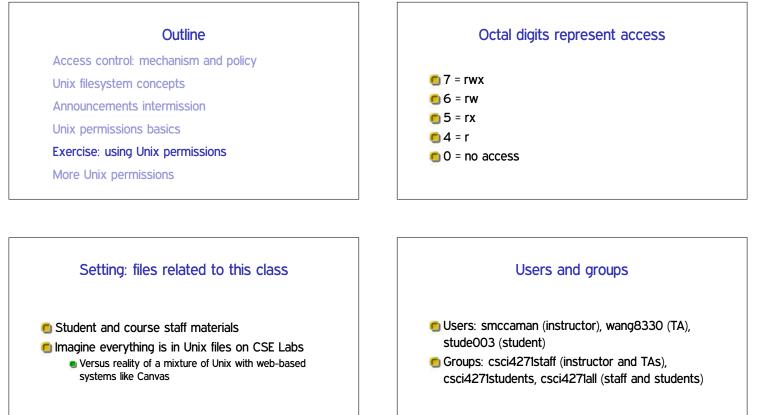
- Core permissions are 9 bits, three groups of three
- Read, write, execute for user, group, other
- 🗐 ls format: rwx r-x r--
- 🖲 Octal format: 0754



Directory mode bits

- Same bits, slightly different interpretation
- 🖲 Read: list contents (e.g., 1s)
- Write: add or delete files
- 🖲 Execute: traverse
- X but not R means: have to know the names





What I want from you

- Brainstorm sets of octal permissions bits that could be used
- For each permission bits set, give user, owner, and file/directory contents/use that would be sensible

Outline

Access control: mechanism and policy Unix filesystem concepts Announcements intermission Unix permissions basics Exercise: using Unix permissions More Unix permissions

Process UIDs and setuid(2)

- UID is inherited by child processes, and an unprivileged process can't change it
- But there are syscalls root can use to change the UID, starting with setuid
- 🖲 E.g., login program, SSH server

Setuid programs, different UIDs

- If 04000 "setuid" bit set, newly exec'd process will take UID of its file owner
 - Other side conditions, like process not traced
- Specifically the effective UID is changed, while the real UID is unchanged
 - Shows who called you, allows switching back

More different UIDs

Two mechanisms for temporary switching:

- Swap real UID and effective UID (BSD)
- Remember saved UID, allow switching to it (System V)

Modern systems support both mechanisms at the same time

Setgid, games

- Setgid bit 02000 mostly analogous to setuid
- But note no supergroup, so UID 0 is still special
- Classic application: setgid games for managing high-score files

Special case: /tmp We'd like to allow anyone to make files in /tmp So, everyone should have write permission But don't want Alice deleting Bob's files Solution: "sticky bit" 01000

Special case: group inheritance

- When using group to manage permissions, want a whole tree to have a single group
- When 02000 bit set, newly created entries with have the parent's group
 (Historic BSD behavior)
- Also, directories will themselves inherit 02000

