Lecture 23 – Access Control

Stephen Checkoway
Oberlin College
Slides based on Bailey's ECE 422

Authentication vs Authorization

- Authentication — Who goes there?
  - Restrictions on who (or what) can access system
- **Authorization** — Are you allowed to do that?
  - Restrictions on actions of authenticated users
- Authorization is a form of **access control**
- Authorization enforced by
  - Access Control Lists
  - Capabilities

Access Control

- Access control is a collection of methods and components that supports
  - confidentiality
  - integrity
- Goal: allow only authorized subjects to access permitted objects
- E.g., Least privilege philosophy
  - A subject is granted permissions needed to accomplish required tasks and nothing more

Subjects access objects

- ***Subject***: active entity that wants to access an object or the data within an object
  - Examples: users and programs
  - must be authenticated (who is the subject?) and authorized (do they have permission to access the object)
- ***Object***: passive entity that contains information
  - Examples: a computer, database, file, computer program, directory, or more fine-grained like a particular field in a table in a database

Example: When a program accesses a file, the program is the subject and the file is the object

Access Control Designs

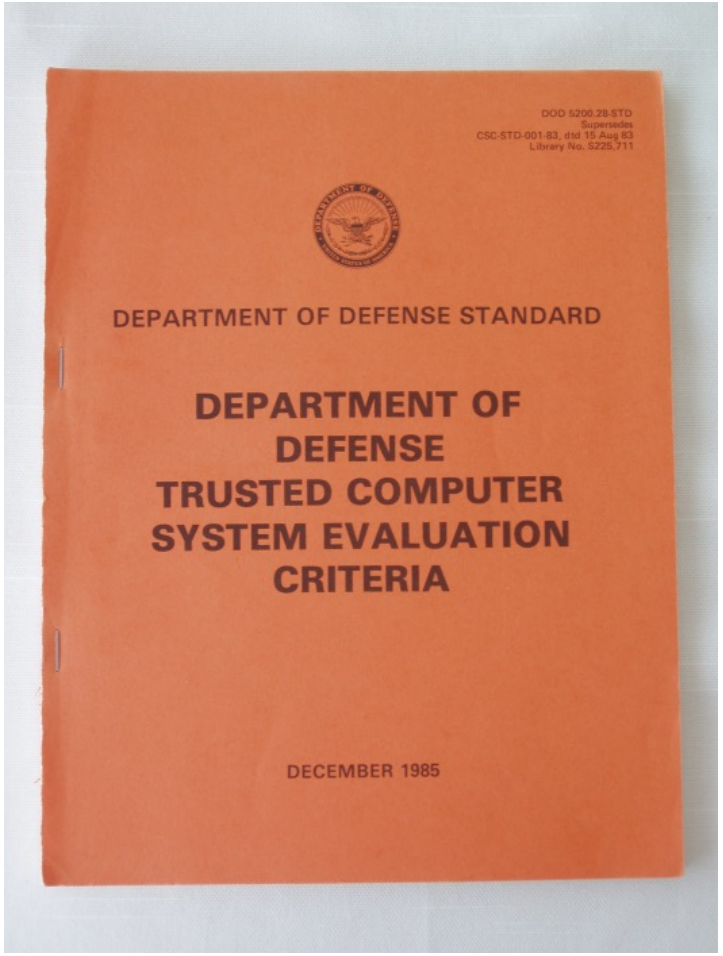
- Access control designs define rules for users accessing files or devices
- Three common access control designs
  - Mandatory access control
  - Discretionary access control
  - Role-based access control

Mandatory Access Control (MAC)

- It is a restrictive scheme that does not allow users to define permissions on files, regardless of ownership
- Instead, security decisions are made by a central policy administrator
- A common implementation is rule-based access control
  - Subject demonstrates need-to-know in addition to proper security clearance
  - Need-to-know indicates that a subject requires access to object to complete a particular task
- Security-Enhanced Linux (SELinux) incorporates MAC

MAC, historically

- Intended to protect classified information
- Subjects and objects have security (or sensitivity) labels
  - Clearance level, e.g., CONFIDENTIAL, SECRET, and TOP SECRET
  - Need-to-know categories
- To access a particular object, the subject needs a clearance level at least as high as the subject's and the need-to-know categories



MACs in OSes

- Windows: Mandatory Integrity Control
  - Four integrity levels: low, medium, high, and system
- Linux: multiple approaches: SELinux, AppArmor, and others
  - Security-Enhanced Linux (SELinux) has fine-grained control (files have labels) but is a pain to administer
  - AppArmor is used in Ubuntu and restricts paths rather than applying labels to files
- macOS/iOS/FreeBSD: Uses TrustedBSD MAC Framework
- Android: Uses SELinux

Discretionary Access Control

- Discretionary access control, or DAC, refers to a scheme where users are given the ability to determine the permissions governing access to their own files.
  - DAC typically features the concept of both users and groups
  - In addition, DAC schemes allow users to grant privileges on resources to other users on the same system.
- Most common design in commercial operating systems
  - Generally less secure than mandatory control
  - Generally easier to implement and more flexible

Common implementation of a DAC

- Users belong to groups
- Every file has an owner (a user) and a group and permissions for the owner, other users in the group, and everyone else
- Every running process is associated with a user and can only read/write/execute files for which it has permission
- Owners of files can change the permissions of those files
  - E.g., chmod on UNIX-like operating systems

Role-Based Access Control

- The role-based access control (RBAC) model can be viewed as an evolution of the notion of group-based permissions in file systems.
- An RBAC system is defined with respect to an organization, such as company, a set of resources, such as documents, print services, and network services, and a set of users, such as employees, suppliers, and customers
- Uses a subject’s role or task to grant or deny object access

Access control mechanism vs policy

- Mechanism of access control (MAC, DAC, RBAC) is distinct from the policy being enforced
- The policy specifies which **subjects** (or groups of subjects or roles) may perform which **actions** on which **objects**

Example of Implementing Policy

File system Access Control

Access Control Entries and Lists

- An Access Control List (ACL) for a resource (e.g., a file or folder) is a list of zero or more Access Control Entries (ACEs)
- An ACE refers specifies that a certain set of accesses (e.g., read, execute and write) to the resources is allowed or denied for a user or group
- Examples of ACEs for folder “CSCI 343 Grades”
  - Professor; Read; Allow
  - Students; Read; Allow
  - Professor; Write; Allow
  - Students; Write; Deny

Linux File System

- Tree of directories (folders)
- Each directory has links to zero or more files or directories
- Hard link
  - From a directory to a file
  - The same file can have hard links from multiple directories, each with its own filename, but all sharing owner, group, and permissions
  - File deleted when no more hard links to it
- Symbolic link (symlink)
  - From a directory to a target file or directory
  - Stores path to target, which is traversed for each access
  - The same file or directory can have multiple symlinks to it
  - Removal of symlink does not affect target
  - Removal of target invalidates (but not removes) symlinks to it

Unix Permissions

- Standard for all UNIXes
- Every file is owned by a user and has an associated group
- Permissions often displayed in compact 10-character notation
- To see permissions, use `ls -l`

```
jk@sphere:~/test$ ls -l
total 0
-rw-r-----  1 jk ugrad 0 2005-10-13 07:18 file1
-rwxrwxrwx  1 jk ugrad 0 2005-10-13 07:18 file2
```

Permissions Examples (Regular Files)

-rw-r--r--	read/write for owner, read-only for everyone else
-rw-r-----	read/write for owner, read-only for group, forbidden to others
-rwx------	read/write/execute for owner, forbidden to everyone else
-r--r--r--	read-only to everyone, including owner
-rwxrwxrwx	read/write/execute to everyone

Permissions for Directories

- Permissions bits interpreted differently for directories
- *Read* bit allows listing names of files in directory, but not their properties like size and permissions
- *Write* bit allows creating and deleting files within the directory
- *Execute* bit allows entering the directory and getting properties of files in the directory
- Lines for directories in `ls -l` output begin with `d`, as below:

```
jk@sphere:~/test$ ls -l
Total 4
drwxr-xr-x  2 jk ugrad 4096 2005-10-13 07:37 dir1
-rw-r--r--  1 jk ugrad    0 2005-10-13 07:18 file1
```

Permissions Examples (Directories)

drwxr-xr-x	all can enter and list the directory, only owner can add/delete files
drwxrwx---	full access to owner and group, forbidden to others
drwx--x---	full access to owner, group can access known filenames in directory, forbidden to others
-rwxrwxrwx	full access to everyone

Special Permission Bits

- Three other permission bits exist
  - Set-user-ID (“suid” or “setuid”) bit
  - Set-group-ID (“sgid” or “setgid”) bit
  - Sticky bit

Set-user-ID

- Set-user-ID (“suid” or “setuid”) bit
  - On executable files, causes the program to run as file owner regardless of who runs it
  - Ignored for everything else
  - In 10-character display, replaces the 4th character (x or -) with s (or S if not also executable)
    - -rwsr-xr-x: setuid, executable by all
    - -rwxr-xr-x: executable by all, but not setuid
    - -rwSr--r--: setuid, but not executable - not useful

Root

- “root” account is a super-user account, like Administrator on Windows
- Multiple root accounts possible
  - Each root account has a user ID of 0
  - This is unusual
- File permissions do not restrict root
- This is *dangerous*, but necessary, and OK with good practices

Becoming Root

- su
  - Changes home directory, PATH, and shell to that of root, but doesn't touch most of environment and doesn't run login scripts
- su -
  - Logs in as root just as if root had done so normally
- sudo <command>
  - Run just one command as root
- sudo -s
  - Runs a shell as root
- su [-] <user>
  - Become another non-root user
  - Root not required to enter password

Setuid bit

- Usually used for programs which need to run as root

```
$ ls -l /usr/bin | grep '^...s'
-rwsr-xr-x 1 root root      72792 May 30  2024 chfn
-rwsr-xr-x 1 root root      44760 May 30  2024 chsh
-rwsr-xr-x 1 root root      39296 Apr  8  2024 fusermount3
-rwsr-xr-x 1 root root      76248 May 30  2024 gpasswd
-rwsr-xr-x 1 root root      47416 May 15  2025 ksu.mit
-rwsr-xr-x 1 root root      51584 Mar  6 16:00 mount
-rwsr-xr-x 1 root root      40664 May 30  2024 newgrp
-rwsr-xr-x 1 root root      64152 May 30  2024 passwd
-rwsr-xr-x 1 root root      55680 Mar  6 16:00 su
-rwsr-xr-x 1 root root     277936 Mar  2 12:56 sudo
-rwsr-xr-x 1 root root      39296 Mar  6 16:00 umount
```

Setgid bit

- Can be used to run a program as a user of a specific group without being a member of the group

```
$ ls -l /usr/bin | grep '^......s'
-rwxr-sr-x 1 root shadow     72184 May 30  2024 chage
-rwxr-sr-x 1 root crontab    39664 Aug 27  2024 crontab
-rwxr-sr-x 1 root shadow     27152 May 30  2024 expiry
-rwxr-sr-x 1 root _ssh      309688 Mar  4 17:55 ssh-agent
```

Sticky bit

- The sticky bit set on a directory changes who is allowed to delete or rename files in the directory
- Normally, file deletion/renaming is controlled by the write permission on the directory
- With the sticky bit set, only the owner of the file (or of the directory or root) can delete/rename a file

```
$ ls -ld /tmp
drwxrwxrwt 19 root root 12288 Mar 30 15:49 /tmp
```

Changing Permissions

- Permissions are changed with chmod or through a GUI
- Only the file owner or root can change permissions
- If a user owns a file, the user can use chgrp to set its group to any group of which the user is a member
- root can change file ownership with chown (and can optionally change group in the same command)
- chown, chmod, and chgrp can take the -R option to recurse through subdirectories

Examples of Changing Permissions

chown -R root dir1	Changes ownership of dir1 and everything within it to root
chmod g+w,o-rwx file1 file2	Adds group write permission to file1 and file2, denying all access to others
chmod -R g=rwX dir1	Adds group read/write permission to dir1 and everything within it, and group execute permission on files or directories where someone has execute permission
chgrp testgrp file1	Sets file1's group to testgrp, if the user is a member of that group
chmod u+s file1	Sets the setuid bit on file1. (Doesn't change execute bit.)

Real and effective UID/GID

- Processes (on UNIX-like systems at least) have both real and effective user IDs and group IDs; UID vs. EUID, GID vs. EGID
- Real UID/GID are the actual user's UID/GID
- EUID/EGID are normally the same as UID/GID but when running setuid/setgid processes, the EUID/EGID is set to the owner/group of the executed file
- File permissions are checked using the EUID/EGID

Changing UID programmatically

- setuid(), seteuid(), setgid(), setegid() change the IDs
- Restrictions on which IDs can be set, read the man pages
- Also Linux has a separate file system user/group IDs (which are almost always the same as the EUID/EGID)
- Also capabilities separate from user/groups impact what ID values can be set

Aside: exploiting setuid root programs

- In project 1, the provided shellcode for most targets was setuid(0)
  execve(“/bin/sh”, NULL, NULL)

- Without changing the real UID to root, sh would set its EUID to the real UID, dropping root privileges