OS Security

An Overview

Instructor: Khaled Diab

What we discussed so far...

- Control-flow Hijacking
  - Buffer overflow
  - Format string vulnerability
  - Integer overflow, Implicit cast, TOCTOU
- Return-to-libc
- Return-oriented Programming
- Defenses

Today’s Lecture

- Security Models
  - Access Control

- UNIX Security Model

System Security

- Three components

- Security Model
  - An abstraction to discuss and decide a policy
  - Recall the Threat Modelling example…

- Security Policy
  - Allowed actions. Who is allowed to do what?

- Security Mechanism
  - Policy implementation
  - E.g., encryption

Security Model

- Subjects (Who)
  - Processes and users
- Objects (What)
  - Memory, files, devices, …
- Operations
  - What do subjects perform to objects?
  - Examples?

Security Policy

- Allowed actions. Who is allowed to do what?

- Examples:
  - Creating a new users
  - Creating new files
  - Reading an existing directory

Access Control Matrix

Access Control Matrix: Example

	File 1	File 2	File 3	File 4
Alice	R	RW	R	RW
Bob	RW	RWX	RWX	RW
Jane	X	RWX	R	RW
John	R	R	R	R

Security Mechanisms

- Two mechanism to enforce a security policy:
  - Access Control List (ACL)
  - Capabilities

- Both are means of access control

Access Control List (ACL)

- Object-oriented approach
- Every object has a list that specifies the what operations subjects can perform
- Each access to an object → requires a check against its list

	File 1
Alice	R
Bob	RW
Jane	X
John	R

Capabilities

- User-centric approach.

- A capability grants a subject permission to perform an action

- A **reference monitor** checks the capability before a subject performs an operation

Access Control

	File 1	File 2	File 3	File 4
Alice	R	RW	R	RW
Bob	RW	RWX	RWX	RW
Jane	X	RWX	R	RW
John	R	R	R	R

Capability List

ACL

ACL vs Capability List

- Efficiency (e.g., when a user makes a request)
  - Capability – just needs the token
  - ACL – need to traverse a list
- Accountability (e.g., who has access to a file?)
  - Capability – needs to look at every user tokens
  - ACL – it's already stored in the list
- Revocation (e.g., revoke access to a resource)
  - Capability – this information is stored in the user catalogue (cannot access)
  - ACL – locate the list of the resources, and remove the access right

Role-based Access Control (RBAC)

- The matrix can get complex as the number of subjects, objects, and operations grow
- **Observation:** Users change more often than roles

	hr/	eng/	admin/	all/
exec	R	R	RW	RW
hr	RW	-	-	R
eng	-	RW	-	R

UNIX

- Started in 1969 at AT&T / Bell Labs
  - Created by Ken Thompson and Dennis Ritchie
- Split into a number of popular branches
  - BSD, System V, Solaris etc.
- Inspired a number of Unix-like systems
  - Linux
- Standardization attempts
  - POSIX…

UNIX Security Model

- Subjects (Who)
  - Users
- Objects (What)
  - Files: sockets, pipes, dev, …
- Operations
  - Read, write, execute

UNIX Groups

- A user may belong to multiple groups
  - Simple RBAC
- Two files to maintain this information:
  - /etc/passwd: primary group
  - /etc/group : additional groups (if any)

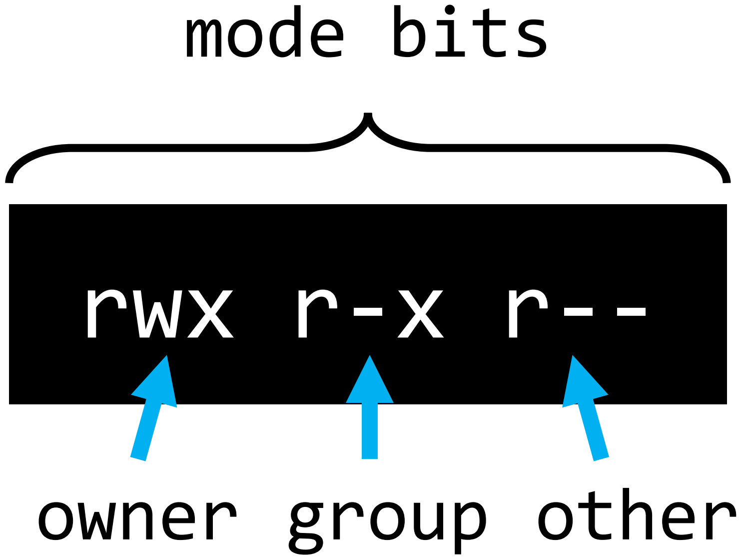
```
sfu@sfu-VirtualBox:~$ groups
sfu adm cdrom sudo dip plugdev lpadmin sambashare
```

UNIX File System Security

- Every file and directory has an owner and group and simple ACL

- File permissions specify what role can do what
  - Three Roles: owner, group, other
  - Three Operations: read, write, execute

- Permissions are set by owner (or root)
  - No delegation



Sticky Bit

- A user can have access to a directory but not to all files in that directory.
  - Example?
- Can the user rename the files that they don't own?

Sticky Bit

- A user can have access to a directory but not to all files in that directory.
  - Example?
- Can the user rename the files that they don't own?

- Sticky Bit:
  - 0: if a user has a permission → can rename/remove files
  - 1: only file owner, directory owner, and root can rename/remove files

```
-rw-r--r--   1 root root     2 Jan 27 19:59 test
drwxrwxrwt  10 root root  4096 Feb  4 18:05 tmp/
```

UNIX File System Security

- Shared resources
  - Potential race conditions
- TOCTOU (discussed earlier)
  - Common race condition problem
- Potential solutions?

UNIX Processes

- Every process has a lot of management info
  - E.g., PID, PPID
  - Scheduling, memory mgmt, etc.
- Real UID (RUID)
  - Which user started the process
- Effective UID (EUID)
  - Determines the permissions for the process
- Saved UID (EUID)
  - The UID to be restored (prior to EUID)

Superuser

- Can do anything!

- UID = 0

- Sys admins assume this role to perform privileged actions
  - Good practice: use the superuser role only when needed

Dropping Privileges

- Login and sshd run as root
  - Authentication
  - Executes a user shell

- But it needs to drop privilege from root to regular user before executing a shell!

Elevating Privileges

- Programs often run with the user ID and group ID of who executed them
- Sometimes, we need to run a program with its owner ID
- Programs have a setuid bit:
  - When set → The EUID becomes the owner ID
  - So that a regular user can perform privileged operations (if owner is root)

```
sfu@sfu-VirtualBox:~$ ll /usr/bin/passwd
-rwsr-xr-x 1 root root 53128 Mar 26  2019 /usr/bin/passwd*
```

- Other Examples?

Shell

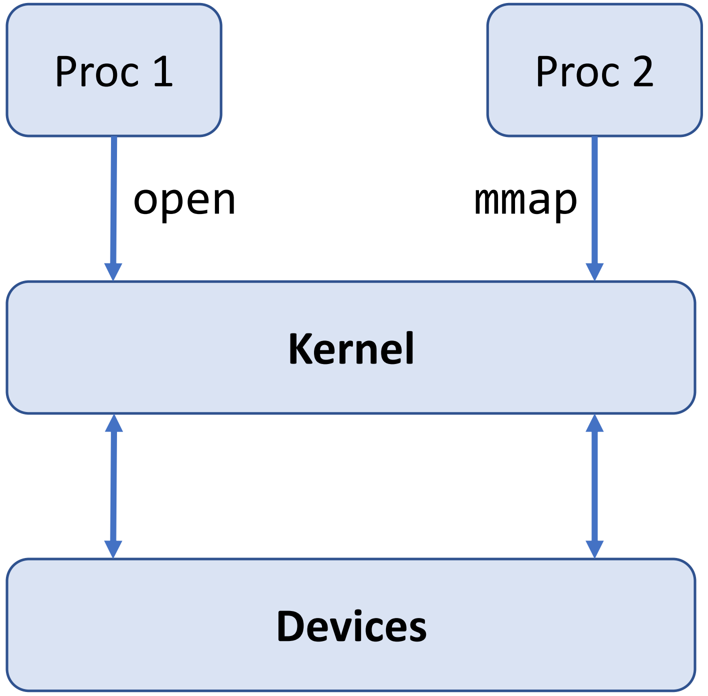
- A core Unix application
- Provides an interface to OS
- Communication between shell and spawned programs via redirection and pipes
- Different flavors: `bash and sh, tcsh and csh, ksh, zsh`

Shell Attacks

- Bugs while parsing commands
  - Can modify or extend shell behavior
  - Effect: user input might become an executable code
  - Recent example?

Kernel Attacks

- Kernel vulnerability
  - usually leads to complete system compromise
  - attacks performed via system calls



Linux Capabilities

- Traditionally: coarse-grained privilege EUID=0 or EUID!=0
  - Can lead to security flaws (e.g., buffer overflow)

- Privileged programs bypass kernel checks

- Towards fine-grained privileges:
  - Divide the superuser role into pieces
  - Assign the program the capabilities it only needs
  - Even when the program is compromised, the damage can be contained

Linux Capabilities: Example

- `CAP_KILL`
  - Bypass permission checks for sending signals
- `CAP_NET_BIND_SERVICE`
  - Bind a socket to privileged ports (port < 1024).
- `CAP_SYS_MODULE`
  - Load and unload kernel modules
- `CAP_SYS_PTRACE`
  - Trace arbitrary processes using `ptrace(2)`

DAC vs MAC

- Discretionary Access Control
  - Example: Linux
  - File owner can set the security policy for objects they own!

- Mandatory Access Control
  - Example: SELinux
  - Centralized component sets the policy if/when needed

Next lecture

- OS-related Attacks
  - Set-UID and Environment variables (tentative)
  - Shellshock
  - Dirty COW