CSE 410/565 Computer Security
Spring 2022

Lecture 13: Operating System Security

Department of Computer Science and Engineering
University at Buffalo

Review

- Previous topics
  - authentication
  - access control
  - database security
  - session establishment and key management

Overview

- The next topics
  - OS access controls and security mechanisms
    - memory protection
    - processor modes
    - authentication and file access control
    - process privileges
  - software security
    - input validation
    - buffer overflow
    - other vulnerabilities

Computer System Components

- Hardware
  - provides basic computing resources, i.e., CPU, memory, I/O devices
- Operating system
  - controls and coordinates the use of the hardware among various application programs
- System and application programs
  - define the ways in which system resources are used to solve computing problems of users
- Users
  - people, machines, other computers

Security Goals of OSs

- Goal 1: enable multiple users to securely share resources
  - separation and sharing of processes, memory, files, devices, etc.
  - what does it involve?
    - memory protection
    - processor modes
    - authentication
    - file access control

Security Goals of OSs

- Goal 2: ensure secure operation in a networked environment
  - what does it involve?
    - authentication
    - access control
    - secure communication (using cryptography)
    - logging and auditing
    - intrusion detection and prevention
    - recovery

Memory Protection

- The operating system enforces access control to memory
- The goal is to ensure that a user's process cannot access other processes' memory
  - fence: hard separation between user and OS space
  - relocation: address adjustment to account for OS space
  - base/bounds registers: start and end of user addresses
  - segmentation: address space separation within a program
  - paging: memory partitioning independent of access decisions
- The operating system and user processes need to have different privileges

CPU Modes

- System mode
  - a.k.a. privileged mode, master mode, supervisor mode, kernel mode
  - can execute any instruction and access any memory location
  - e.g., accessing hardware devices, enabling and disabling interrupts, accessing memory management units, modifying registers, etc.
- User mode
  - access to memory is limited, some instructions cannot be executed
  - e.g., cannot disable interrupts, arbitrarily change processor state, access memory management units, etc.
- Transition from user mode to system mode must be done through well defined call gates (system calls)

Operating System Protection

- System calls are guarded gates from user mode into kernel mode
  - they use a special CPU instruction (often an interrupt) to transfer control to a predefined entry point in more privileged code
  - they allow to specify where the more privileged code will be entered and the processor state at the time of the entry
  - privileged code examines the processor state and/or the stack set by less privileged code and determines whether to allow the request
- Part of the OS runs in the kernel mode (OS kernel)
- Other parts of the OS run in user mode, including service programs and user applications, as processes
- Superuser (or root) privileges are different from kernel mode rights

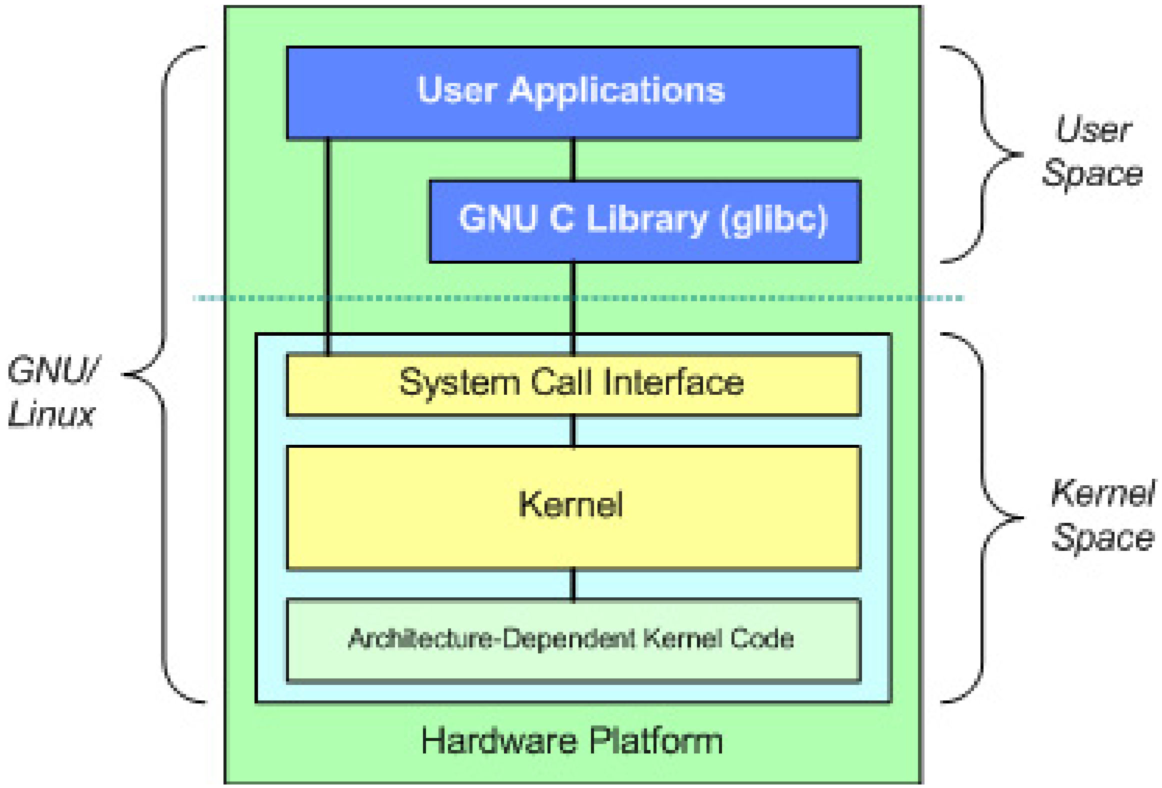
Types of Kernels

- Monolithic kernel
  - one big kernel provides all services
  - e.g., file system, network services, device drivers, etc.
  - all kernel code is run in one address space
  - different services directly affect each other
  - example: Unix variants
    - Linux kernel had over 25 million lines of code in 2018
    - advantages: efficiency
    - disadvantages: complexity, bugs in one part affect the entire kernel
  - kernels with loadable kernel modules are still monolithic

Types of Kernels

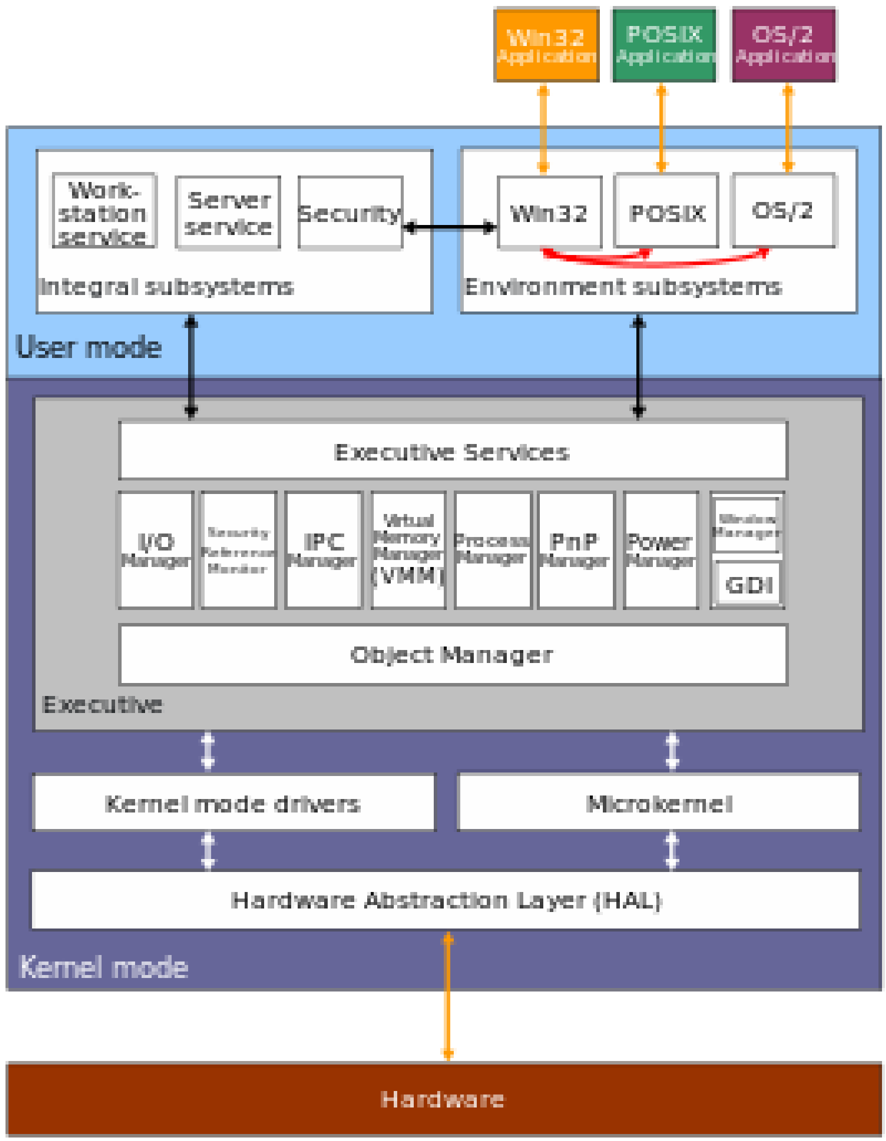
- Microkernel
  - minimal kernel that provides only the mechanisms needed to implement OS services
  - e.g., low-level address space management, threat management, and inter-process communication (IPC)
  - operating system services are provided by user-mode servers
    - these include device drivers, protocol stacks, file systems and user-interface code
  - advantages: better achieves the least privilege, can tolerate failures/errors in device drivers, etc.
  - disadvantages: performance, failure in key OS services still brings the system down

Example Architectures: Linux



Source: Wikipedia

Example Architectures: Windows NT



Source: Wikipedia

Security Mechanisms in User Space

- Types of security mechanisms
  - authentication
  - access control
  - logging and auditing
    - record system information to a log
    - what should we record?
      - full range of data and events, normal and suspicious
      - e.g., every logon attempt, permission changes, network connection events, system calls, access to selected applications, system management events
    - how can we record such data?

Security Mechanisms in User Space

- Types of security mechanisms (cont.)
  - logging and auditing
    - recording can be done at system level, application level, and user level
    - examples include application logging, system call interception, packet sniffing, etc.
    - audit trails must be protected!
      - restricted access to the trails
      - backing up to a different system
      - enforcing write-only or write-once mechanisms

Security Mechanisms in User Space

- Types of security mechanisms (cont.)
  - intrusion detection and prevention
    - detect and report possible network and computer system intrusion or attacks
    - passive intrusion detection provides only detection
    - reactive intrusion detection provides intrusion prevention
    - types of intrusion detection systems (IDSs) vary, e.g., host-based, network-based, etc.
  - recovery
    - if a break-in is detected, investigate the cause and assess the damage
    - bring the system to a stable state

Permissions in Unix

- Recall that we are dealing with uid and gid permissions
- Processes are subjects
  - associated with uid/gid pairs such as (ruid, rgid), (euid, egid), (suid, sgid)
- Objects are files
  - 12 permission bits
    - read/write/execute for user, group, and others
    - suid, sgid, sticky
- There are associated system calls for read, write and execute operations

Permissions in Unix

- Process uid model in modern Unix systems
  - each process has three user IDs
    - real user ID – owner of the process
    - effective user ID – used in most access control decisions
    - saved user ID – supported only on some systems
  - similarly, there are three group IDs
    - real group ID
    - effective group ID
    - saved group ID

Permissions in Unix

- What effect do suid, sgid, and sticky bits of files have?

	suid	sgid	sticky bit
non-executable files	no effect	affect locking	not used any more
executable files	change euid when executing the file	change egid when executing the file	not used any more
directories	no effect	new files inherit the group of the directory	only the owner of the file can delete

- suid and sgid allow executables to inherit the uid and gid privileges, respectively, of the file owner when executed

Permissions in Unix

- When a process is created by using `fork`
  - it inherits all three user IDs from its parent
- When a process executes a file using `exec`
  - it inherits three user IDs unless the suid bit of the file is set
  - if the suid bit is set, the effective uid and saved uid are assigned the user ID of the file owner
- Why do we need suid/sgid bits?
  - some operations require higher (superuser) privileges than a process can have
    - e.g., halting the system, listening on privileged ports (TCP/UDP port below 1024), etc.

Permissions in Unix

- Why do we need suid/sgid bits?
  - some operations are not modeled as files
  - system integrity requires not only controlling who can write, but also how it is written
  - file level access control is not fine-grained enough
- Are there security implications of having programs with suid/sgid?
  - setuid programs are typically setuid root
  - this violates the least privilege principle
  - why is it bad?
  - how can an attacker exploit this problem?

Permissions in Unix

- Is there a way to make setuid programs safer?
- An existing solution is to change effective user IDs
  - a process that executes a setuid program can drop its privilege
  - there are two possibilities
    - drop privilege permanently
      - removes the privileged uid from all three user IDs
    - drop privilege temporarily
      - removes the privileged uid from it effective uid, but stores it in its saved id
      - later the process may restore privilege by restoring privileged uid in its effective uid

Permissions in Unix

- Early Unix systems
  - there were two user IDs: real uid and effective uid
  - privileges could be dropped only permanently
- Later Unix systems
  - saved uid was introduced
  - privileges could be dropped temporarily
- There are different implementations with different system calls
  - can be inconsistent, incompatible

Computer Break-Ins

- What happens in a typical computer break-in?
  - get your foot in the door
    - steal a password file and run dictionary attack
    - sniff password off the network or through social engineering
    - use input vulnerability in network-facing programs
      - e.g., web server, mail server, browser, etc.
  - use partial access to gain superuser privileges
    - break some mechanism on the system
    - often this means exploiting vulnerabilities in some local programs

Computer Break-Ins

- Steps involved in a typical computer break-in (cont.)
  - setup some way to return
    - install login program or web server with a back door
  - cover your tracks
    - disable intrusion detection, virus protection, system functions that show list of running programs, ...
  - perform desired attacks
    - break into other machines
    - take over the machine
    - ...

OS Hardening

- It is critical to setup the OS with adequate security and maintain it
  - initial setup and patching
  - removing unnecessary services and applications
  - configuring users, groups, authentication
  - configuring permissions
  - installing additional security controls
  - testing the system security

Summary

- Operating systems security covers
  - memory protection
  - restricting access to critical resources
  - controlling process privileges
- Software security is next
  - it is a large topic that covers different types of attacks
  - buffer overflow is one of the most common software vulnerabilities
  - we'll also look at safe code writing practices