COMP5900 OS Security

William Findlay

January 16, 2020

1 Introduction

- trusted computing base
  - applications that are essential to functioning of the OS
  - e.g., passwd
  - these would probably be okay to talk about for OS vuln. but kernelspace code is preferred

1.1 Bloom's Taxonomy

- course targets top 3 sections (for evaluation)
  - create
  - evaluate
  - analyze
  - (some understanding)

1.2 What is an OS?

- kernel
- essential applications (systemd, passwd, etc.)
- what does it do?
  - scheduling
  - network stack
  - file systems
  - block I/O on disk
  - hardware interrupts (e.g. I/O)
  - at least basic access control (memory protection, etc.)
  - often runs in supervisor mode (apparently not necessarily? But I don't agree with this...)

1.3 What is the Most Secure OS?

- probably something task-specific

1.4 Group Activity: Come up with an OS-less implementation for a word processor

- interrupt handling for keyboard
  - need at least some basic scheduler that can pause and resume main execution
- interface with monitor for graphical display
- block I/O driver for disk
  - filesystem to organize data
- hmm... this is starting to feel like we just implemented our own task-specific OS
  - that's the main takeaway here!

2 Secure OS

2.1 Van Oorschot Chapter 5.0 - 5.2

2.1.1 Intro

- early security had same challenges we face today
  - protecting programs from others
  - restricting access to resources
  - "protection" means mostly memory access control
- memory is important
  - holds data
  - holds programs
  - I/O devices through memory address and files
  - files -> both main memory and secondary storage
- early protection
  - virtual addresses
  - access control lists
  - limited process address space
  - these fundamentals are still used today
- Multics
  - security very influential in its early design
  - original UNIX was heavily based on Multics

2.1.2 Memory protection, supervisor mode, and accountability

- batch processing
  - prepare jobs ahead of time and submit them together as a batch job
- time-sharing systems
  - allowed shared use of a single computer
  - (preferable to batch jobs from a usability standpoint)
  - same way single-user computers work today with one user running many programs
- resource conflicts
  - processes running simultaneously can try to access the same resources

    - intentionally or otherwise
    - if a program could access full memory of the machine, errors could corrupt OS data or code
- supervisor
    - runs with higher permissions in the protection CPU (ring 0, 1, 2)
    - no other program can alter the privileged bit
    - a special machine instruction immediately transfers control to the supervisor
- privileged bit
    - process is running in supervisor mode
- descriptor register
    - holds a memory descriptor that describes base and upper bound
    - lowest addressable memory by a process and a number of words from that point that are addressable
- limitations of memory-range based protection
    - all-or-nothing mode of control
    - either you have full access or no access
    - allows full isolation, but not fine-grained sharing
- segment addressing with access permissions
    - segment = collection of words representing a logical unit of information
    - descriptor segment per process maintained by OS
        - holds segment descriptors that define addressable memory and permissions
    - descriptor base register points to memory descriptor of active process
- permissions on virtual segments
    - R non-supervisor can read
    - W can be written to
    - X can be executed
    - M run in supervisor mode (if X)
    - F all access attempts trap to supervisor
    - now the same physical segment can be given different access for different processes
- accountability, UIDs, and principals
    - UID (maps users to a unique identifier)
    - "principal" -> abstracts the entity responsible for code execution from the actual user or program actions
    - UID is the primary basis for granting permissions
- roles
    - assign distinct UIDs to distinct privileges
    - should follow principle of least privilege

2.1.3 Reference monitor, access matrix, security kernel

- reference monitor
    - concept that "all references by any program to any other program, data, or device are validated"
    - conceptualized as one reference monitor, but in practice, would be a lot of reference monitors working together

- access matrix
  - 2D matrix of subjects, objects
  - taking a row (subject) gives a capabilities list
  - taking a column (object) gives an access control list
  - each intersection in this matrix defines a set of permissions
- security kernel
  - reference validation
  - audit trails via audit logs (user X did Y at time Z)
    - these might not necessarily need to be tamper-proof, depends on needs
  - needs to be:
    - tamper-proof
    - always invoked (not circumventable)
    - verifiable (needs to be minimal / small enough to make this possible)
- protection mechanisms
  - ticket-oriented (capabilities)
    - access token allows entry to an event, as long as ticket is authentic
  - id-based
    - authorization lists based on ID

2.2 Jaeger Chapter 1

2.2.1 Secure OS

2.2.2

2.3 Jaeger Chapter 2