COMP5900 OS Security

William Findlay

January 18, 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

- general-purpose -> complex
- task-specific -> not so complex
- general purpose OS are hard to secure because of their complexity
- ensuring security depends on securing
  - resource mechanisms
  - scheduling mechanisms

2.2.1 Secure OS

- enforce security goals despite the threats faced by the system
  - implement security mechanisms to do this
- secure OS possible?
  - probably not
  - a modern OS by definition can probably never be 100% secure
  - security as a negative goal
- understanding secure OS requires understanding
  - security goals
  - trust model
  - threat model

2.2.2 Security Goals

- define operations that can be executed by a system while remaining in a secure state
  - i.e. prevent unauthorized access

- high level of abstraction
- define a requirement that the system's design can then satisfy
- we want to maintain: secrecy, integrity, availability
  - secrecy = limit read access for objects by subjects
  - integrity = limit the write access for objects by subjects
  - availability = limit the resources that a subject may consume (i.e. no DoS)
- subjects
  - users, processes, etc.
- objects
  - resources of the system that subjects may or may not access in various ways
  - e.g. files, sockets, memory
- security goals can be
  - defined by function (e.g. principle of least privilege)
  - defined by requirements (e.g. simple-security property)

2.2.3 Trust Model

- trust model
  - defines the set of software and data we trust to help us enforce our security goals
  - we depend on this model to correctly enforce our security goals
- trusted computing base
  - trust model for an operating system
- TCB should **ideally** be minimal to the extent that we require
  - in practice, this is a wide variety of software
- TCB includes
  - all OS code (assuming no boundaries as in a monolithic kernel)
  - other software that defines our security goals
  - other software that enforces our security goals
  - software that bootstraps the above
  - software like Xorg that performs actions on behalf of all other processes
- a secure OS developer needs to prove their system has a viable trust model
  (1) TCB must mediate all sensitive operations
  (2) verification of the TCB software and data
  (3) verification of TCB tamper-resistance
- identifying and verifying TCB is a complex and non-trivial task

2.2.4 Threat Model

- defines a set of operations that an attacker may use to compromise the system
- assume a powerful attacker who
  - can inject operations from the network
  - may be in control of non-TCB applications
- if the attacker finds a vulnerability that violates secrecy or integrity goals, the system is compromised
- highlights a critical weakness in commercial OSes

    - assume that all software running on behalf of a subject is trusted by the subject
- our task? protect the TCB from threats
    - easier said than done
    - user interacts with a variety of processes
    - users are untrusted
    - TCB interacts with a variety of untrusted processes

2.3 Jaeger Chapter 2

2.3.1 Protection System

- protection system consists of
    - protection state
    - protection state operations
- protection state
    - what operations can subjects perform on objects
- protection state operations
    - what operations can modify the protections state
    - (this is distinct from the operations that the protection state describes)

Lampson's Access Matrix.

- protection state
    - rows = subjects
    - cols = objects
    - select row -> capability list
    - select col -> access control list
    - each entry specified privileges subject -> object
- protection state operations
    - determine which processes can modify cells

Mandatory Protection Systems.

- we don't want untrusted processes tampering with the protection system's state by adding subjects, objects, operations
- discretionary access control system (DAC)
    - an access control system that permits untrusted modification
    - *safety problem*
        - how do we ensure that all possible states deriving from initial state will not provide unauthorized access
- mandatory protection systems / mandatory access control (MAC)
    - protection system can only be modified by trusted administrators via trusted software
    - mandatory protection state -> subjects and objects are represented by labels
        - state describes operations subject labels -> object labels
    - labeling state

        - state for mapping subjects and objects to labels
    - transition state
        - describes legal ways subjects and objects may be relabeled
- set of labels being fixed in MAC doesn't mean that set of subjects/objects are fixed
    - we can dynamically assign labels to created subjects and objects (labeling state)
    - we can dynamically relabel subjects and objects/resources (transition state)

2.3.2 Reference Monitor

- classical access enforcement mechanism
- takes request as input
- outputs binary response -> is the request authorized or not?
- main components?
    - interface
    - authorization module
    - policy store

Reference Monitor Interface.

- defines queries to the reference monitor
- provides an interface for checking security-sensitive operations
    - (security-sensitive means it may violate security policy)
- e.g., consider the `open` system call in UNIX (reference monitor decides what is allowed / disallowed)

Authorization Module.

- takes interface inputs, converts to a query for the policy store
- this query is used to check authorization
- authorization module needs to map PID to subject label and object references to an object label
- needs to determine the actual operation(s) to authorize

Policy Store.

- database that holds protection state, labeling state, transition state
- answers queries from the authorization module
- has specialized queries for each of the three states

2.3.3 Secure Operating System Definition

- a secure operating system's access enforcement satisfies the reference monitor model
- the reference monitor model defines the necessary and sufficient properties of a system that securely enforces MAC
- three guarantees:
    (1) complete mediation -> ensure access enforcement for all security-sensitive operations

(2) tamper proof -> cannot be tampered with from outside the TCB (untrusted processes)
(3) verifiable -> small enough to be subject to testing, analysis

2.3.4 Assessment Criteria

Complete Mediation.

- how does the reference monitor interface ensure that all security-sensitive operations are mediated correctly
- does the reference monitor mediate security-sensitive operations on all system resources
- how do we verify complete mediation?

Tamper Proof.

- how does the system protect the reference monitor and its protection system from modification?
- does the protection system protect TCB programs?

Verifiable.

- what is the basis for TCB correctness?
- does the protection system enforce security goals?

3 Multics

3.1 Jaeger Chapter 3

3.1.1 Jaeger 3.1

3.1.2 Jaeger 3.2

3.1.3 Jaeger 3.3

3.2 Virtual Memory in Multics (Video)

Generating Address.

- 36 bits in total
  - 18 bit segment number
  - 18 bit word number
- i.e. 2^{18} segments and within each segment 2^{18} words

Instruction Format.

- segment tag
  - has segment number and word number
- address

- operation code
- external flag
- addressing mode (2 bits)
    - corresponds to one of four pointers
        1. argument pointer
        2. base pointer (bottom of stack frame)
        3. linkage pointer
        4. stack pointer (top of stack [frame])

Segment Number.

- can only be changed by ring 0 (supervisor mode) program

Word Number.

- can be changed by user mode programs

Switching Processes.

- all the kernel does is substitute a new descriptor base register (with a different segment number)

Apple II and Stuff (1982).

- introduced copy protection on memory / disks
- no virtual memory (they didn't need it, hobbyists didn't need to care about security)

3.3 Protection in an information processing utility (Graham 1968)

Introduction.

The Environment.

Why Protection?.

Properties of Satisfactory Protection Mechanisms.

The Abstract Model.

Software Support.

Additional Complexities in the Information Processing Utility.