COMP5900 OS Security

William Findlay

January 25, 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 Defining a 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?

2.4 Class Review

Other Verifiability Techniques.

- code review
- regression (TDD)
- formal verification
- fuzzing

3 Multics

3.1 Jaeger Chapter 3

- Multics = the first modern OS
- invented many fundamental OS concepts
  - segmented virtual memory
  - shared memory for multiprocessors
  - hierarchical filesystems
  - online reconfiguration
  - (many others)
- invented many fundamental *secure OS* concepts
  - reference monitor

- ring protection model
- protection systems
- protection domain transactions
- multilevel security policies
- (many others)

3.1.1 Jaeger 3.1

- Multics emerged from the CTSS (Compatible Timesharing System) system
- early timesharing systems could support a few jobs at a time
  - vs batch processing could only do one at a time
- Project MAC (Multi-Access Computer) aims to create a general-purpose timesharing service to support large number of users simultaneously
  - this would require several functions such as multiplexing of devices for different processes, scheduling processes, communications between processes, and protection from other processes
- IBM wasn't interested in the idea, so GE took over
- although it was considered a failure (bigger, slower, not reliable), it brought important OS and security features that would lead to UNIX
- Multics cost around $7 million initially (only 80 licenses sold)
- last department to use Multics was the Canadian Department of Defense (LOL we got issues boi)

3.1.2 Jaeger 3.2

- layered architecture
- resources organized hierarchically

Multics Fundamentals.

- processes are executables (they run program code)
- all code, data, I/O devices,etc that processes have access to are called segments
- hierarchy of directories that may contain
  - sub-directories
  - segments
- a process's protection domain defines the segments it can access and what operations it can perform on them
- segments can be stored in a process's context or secondary storage
- each process has its own descriptor segment which contains segment descriptor words that point to the segments the process has access to.
- if the segment is not in the descriptor segment, it must name the segment (like a file path)
  - if the process has permission, the segment is added to its descriptor segment
- descriptor segment has a set of *segment descriptor words* that refer to all directly accessible segments

Multics Security Fundamentals.

- *answering service* process implements the login system
  - to authenticate the user, answering service loads password SDW into its own descriptor segment
  - Multics supervisor must authorize this
- *supervisor* implements all security-critical functionality such as authorization, segmentation, I/O, scheduling, etc.
- *protection rings* protect the supervisor from other processes
  - the rings form a hierarchy with ring 0 as the most-privileged
  - the supervisor's segments are assigned to ring 0 and 1
  - higher rings cannot access lower rings
- if the user and password match, the answering service creates a user process with the appropriate code and data segments for that user
- each live process segment is accessed by the SDW
- the SDW contains
  - the address of the segment in memory
  - its length
  - ring brackets
  - process's permissions
  - number of gates for the segment (code segments only)
- ring brackets
  - define access according to process rings
  - code segment also has a call bracket that defines whether it can be run
  - also define access to protection domain transition rules

Multics Protection System Models.

Access Control Lists (ACLs).

- each ACL entry specifies the process with a user identity with operations that it can perform on this object.
- segments can have r,w,e permissions
- directories can have r,w,e,s,m,a
- segments ACL are stored in its parent's directory
  - this means that checking for permission or any modification to segment ACL is done through the parent directory
- if the user with the process has permission for operation on the object, then the reference monitor authorizes the creation of SDW with those permissions.

Rings and Brackets.

- aside from ACL, Multics limits access based on protection rings as well
- each segment contains a ring bracket that has r,w,e permission of processes on that segment
  - a segment's bracket defines the ranges of ring that can have certain permission

(rwe) to the segment
- suppose a process from ring r wants to access a segment with an access brackets of $(r1, r2)$ and we have these rules:
  - if $r < r1$, then the process can read and write to the segment
  - if $r1 \leq r \leq r2$, then the process can read the segment only
  - if $r2 < r$, then the process has no access to the segment
- the above rules ensure that lower rings are more privileged and has more access to segments than higher rings
- call brackets define execute permissions on code segments as well as transition states
  - can freely transition to lower rings
  - segment gates control transitions into higher rings

Multilevel Security.

- each directory stores a mapping from each segment to a secrecy level
- Multics also stores an association between each process and its secrecy level
- operations are authorized as follows:
  - writes require a segment to have segment secrecy $\geq$ process secrecy
  - reads require a segment to have segment secrecy $\leq$ process secrecy
  - read/writes require segment secrecy = process secrecy
- MLS prevents information leakage

Multics Protection System.

- ACL, Ring Brackets, and MLS are all taken together
  - ACL and Ring Brackets are discretionary, while MLS is mandatory
- examples
  - requested operation is READ:
    - ACL checks if user has read access
    - MLS policy checks to verify that the object's secrecy level is dominated by or equal to the process
    - the access bracket is checked to ensure the process has access to the object's segment
  - requested operation is WRITE:
    - ACL checks if user has write access
    - MLS policy checks to verify that the object's secrecy level dominates or equal to the process
    - the access bracket must allow the current ring write access
  - requested operation is EXECUTE:
    - similar to write operation
    - the process must have execute permission in the segment's ACL
    - MLS policy must allow for reading the segment
    - call bracket is used instead of access bracket, and call bracket must allow execution
    - the request may result in protection domain transition

    - transition from process's current ring r to the ring specified by the segment in the call bracket r'
    - when a process invokes a code segment with a call bracket with r < r1:
        - the process must transition to r'(a lower privileged ring)
    - when a process invokes a code segment with a call bracket where r2 <= r <= r3:
        - the process transitions to r' by using one of the gates in the code segment as entry point

Multics Reference Monitor.

- Multics' reference monitor is implemented by the supervisor
- each Multics instruction either accesses a segment via a directory or via a SDW, so authorization is performed on each instruction
- the supervisor also performs protection domain transitions as mentioned above
- a transition that requires to go through a gate segment needs to verify the following:
    - the number of arguments expected
    - the data type on each argument
    - access requirement for each argument (read, read/write)
- the gate segment is also known as the gatekeeper
- when we return from the more privileged ring to a lower privileged ring, we need to ensure that no information is leaked.
    - the supervisor copies arguments from its segment to another segment (accessible to the called procedure).
        - copying is necessary because we don't want unauthorized access from the calling procedure
        - the caller must also be careful not to copy unauthorized information (private keys), since the less-privileged code may be able to use to impersonate the higher-privileged code
    - Multics allows the caller to provide a gate for return (return gate).
        - multiple calls can result in a stack of return gates, so SDW is not suitable for return gates
        - the supervisor must maintain this stack of return gates for each process.
- the fundamentals of the reference monitor is implemented in ring 0
    - other utility such as file system search utility are in ring 1. determination of a directory or segment from a name is performed there, but authorization of whether this access is permitted is done in ring 0

3.1.3 Jaeger 3.3

Complete Mediation.

- how does the reference monitor interface ensure that all security-sensitive operations are mediated correctly?
    - Multics provides complete mediation at the segment level.
- does the reference monitor interface mediate security-sensitive operations on all system

resources?
  - Multics mediates each segment access at the instruction level,Multics mediates memory access completely. Multics also mediates ring transitions, in both directions. Thus, the reference monitor provides mediation at memory and ring levels.
- how do we verify that the reference monitor interface provides complete mediation?
  - to verify complete mediation, we need to verify that ring transitions and segment accesses are mediated correctly. These operations are well-defined, so it is straightforward to determine that mediation occurs at these operations.

Tamper-proof.

- how does the system protect the reference monitor, including its protection system from modification?
  - Multics' reference monitor is implemented by ring 0 procedures. Ring 0 procedures are protected by a combination of protection ring isolations and system-defined ring bracket policy. The ring bracket policy prevent processes outside of ring 0 from reading or writing reference monitor code or state directly.
- does the protection system protect all the trusted computing base programs?
  - Multics TCB consists of the supervisor (ring 0) and from ring 1 to ring 3. Ring 4 and above are standard user processing. Rings 0 to 3 can be considered part of the TCB.
  - discretionary nature of ring protection makes this difficult
  - if we compromise one process in TCB, it can undo all ring protection at its ring level
  - we can say that the TCB is securable, but that its tamper resistance is brittle

Verifiable.

- what is basis for the correctness of the system's trusted computing base?
  - the implementation of the Multics TCB is too large to be formally verified. This goal was not achieved even though they did try to aim to minimize the implementation as much as possible
- does the protection system enforce the system's security goals?
  - protection state: MLS secrecy protection is enforced as long as the TCB is not compromised
  - labeling state: Verifying the correct labeling of segments and processes is challenging, since most of the labeling is done manually. The Multics policy does not explicitly state how new processes and segments are labeled
  - transition state: Permits code from a less-privileged ring to transfer control to code in a more-privileged ring through either gates or return gates based on the call bracket rules. The security of these transitions depends on the correctness of the gates

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)

Challenges of protecting an IPU.

- lots of users
- many users using the system simultaneously
- more than one CPU, running multiple programs at a time

- sharing and not sharing
  - system could be used for applications that have sensitive data
  - system could also be used by applications that do want to share their memory with others (maybe on parts of it)

Why do we need protection?.

- protection is not strictly necessary for single-user applications, but still desirable
  - aid debugging
  - prevent application errors from propagating (good even in the benign case)
  - (localizes the source of the original error)
- more than one user makes things much more interesting
  - even simplest multi-user systems share the supervisor program (i.e. reference monitor state) between all users
  - we need to protect this from being tampered with
  - user information stored on multi-user system must be protected for privacy reasons

3.3.1 Defining Satisfactory and Unsatisfactory Protection Mechanisms

Protection Mechanisms Before Multics (Unsatisfactory).

- mode switch for supervisor instructions (paper calls them master and slave)
- memory bounds register (descriptor register)
- what is included in privileged instructions?
  - I/O instructions
  - changing mode switch
  - changing contents of memory bounds register
- mode switch can protect information on storage media that is not actively being looked at
  - but it cannot protect data in active memory... we need the memory bounds register for that
- what was wrong with the original solution?
  - all or nothing
  - either have all privileges, or have none
  - **unsatisfactory** for an IPU

Satisfactory Protection Mechanism Properties.

- isolation
  - should be possible to completely isolate one process from another
- sharing
  - should be easy and convenient to select which parts of a process should be shared with another process
- layering
  - single process layers of protection should be available to system and user
  - "need to know" principle should be easily applicable to a reasonable degree

- calling procedures across layers
  - should be do-able without any special programming

3.3.2 The Abstract Model

Segment Descriptor (Isolation, Sharing).

- pages are invisible to the user (managed by hardware)
- segments contain words of arbitrary length stored in pages
- addressing is done by (S, W) where S is segment number and W is word number
- descriptor needs to have
  - beginning of segment
  - length
  - access indicator
  - (ring number)
- every processor has exactly one segment descriptor

Location Counter.

- keeps track of next available location for memory assignment in current segment
- needs to have
  - (ring number)
  - procedure segment number
  - word number

Protection Rings (Layering).

- up to m rings
- ring 0 is most privileged, ring m is least privileged
- every segment is assigned to one and only one ring
- a process running in ring i has no access to any segment in ring $j < i$

4 Unix

4.1 Paul Chapter 5.4 Setuid, Setgid, RUID, EUID, SUID

- Setuid bit:
  - when a process runs a program with this bit set, the process's userid will be set to the owner's userid so that the calling process can have more resources
  - `s` means executable with setuid
  - `S` means setuid bit but not executable
- Setgid:
  - analogous to setuid, but applies to groups
- Real UID (RUID):
  - process' owner
- Effective UID (EUID):

    - determines privileges on resource access requests, changes to allow non-privileged user to access files (owned by root)
- Saved UID (SUID):
    - saves UID when a process needs to lower its privileges temporarily
    - EUID is saved as SUID and then changed
    - it can return its EUID to SUID later
- PID:
    - when a process forks, it creates another process with the same parent UID triple (RUID, EUID, SUID)
    - PID is new for child process
    - kernel version = TGID (thread group ID)
- TID
    - same as PID but for threads
    - kernel version = PID (not to be confused with userspace PID, which is TGID)

4.2 Paul Chapter 5.5 Directory permissions and inode-based example

- each directory has a list of entries structured as follows:
    - dir-entry = (d-name, d-inode), where d-name is directory name and d-inode is the file's inode
- directory permissions:
    - R: list contents
    - W: edit contents, however X is also needed to rename and delete files
    - X: traverse and search, allows access to inode meta-data
    - setuid: no meaning (used to mean something historically)
    - setgid: group value assigned to group created the dir-entry
    - t-bit(text/sticky bit): prevents deleting or renaming of other people's files within dir. Root and owner have all controls
- world-writable files:
    - If second last bit is w then it is world-writable
- figure 5.6 in Paul's book has a great pic for dir structured
- a common system default for directories is 777 (default mask 022)
    - results in 755 (rwx for user, rx for others and groups)