CS 563 - Advanced Computer Security: Foundations II

Professor Adam Bates
Fall 2018

Administrative

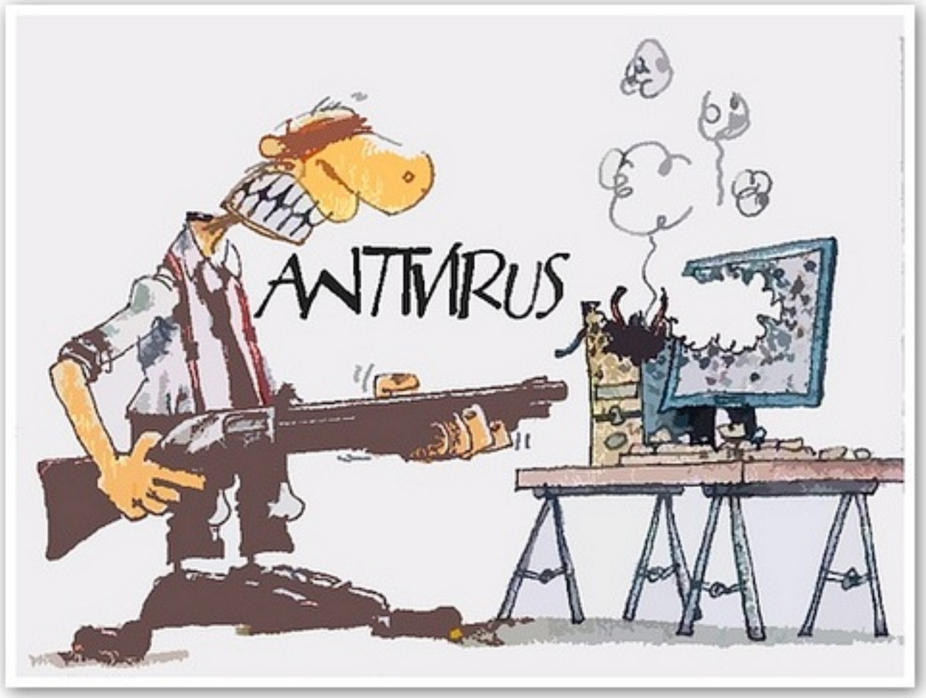
Learning Objectives:
- Explore how the security of Multics failed in practice
- Understand SCOMP and contrast its features to other operating systems (past and present)

Announcements:
- E-Ink tablets approved for class use
- Reaction paper was due today (and all classes)
- Feedback for reaction papers soon

Reminder: Please put away (backlit) devices at the start of class

Today

was multics secure?



Multics: Ten Years in…

- 1964: Multics project conceived as a collaboration between MIT, General Electric, Bell Labs
- 1965: 6 papers on Multics are published at the Fall Joint Computer Conference (we read one of them).
- 1965: Early versions of Multics launch
- 1969: MIT's Multics deployment made publicly available to paying customers; hundreds of accounts created.
- 1970: Second Multics deployment commissioned by Air Force at the Rome Air Development Center (RADC).
- 1972: Karger and Schell begin vulnerability analysis, finalize this report in 1974.

Vulnerability Analysis

- Evaluation of Multics system security 1972-1973
  - Roger Schell and Paul Karger
  - Schell: security kernel architecture, GEMSOS; architect of Orange Book
  - Karger: capability systems, covert channels, virtual machine monitors
- Criteria: Multics is "securable" (1.3.3)
  - Based on security descriptor mediation
  - Ring protection

Vulnerability Analysis

- Criteria details
  - Look for Multics vulnerabilities
  - Is reference monitor practical for Multics?
  - Identify necessary security enhancements
  - Determine scope of a certification effort
- Logistics
  - Used MIT and RADC deployments
  - Honeywell 645 running a Multics system (old HW)
  - Limited Time: find one vulnerability per area, "not exhaustive or systematic

Results Overview

- Design is sound, implementation is ad hoc
- One or more vulnerabilities uncovered at each of 3 layers:
    1. hardware
    2. software
    3. procedure
- Vulnerabilities discovered at RADC, weaponized and validated against the MIT deployment.

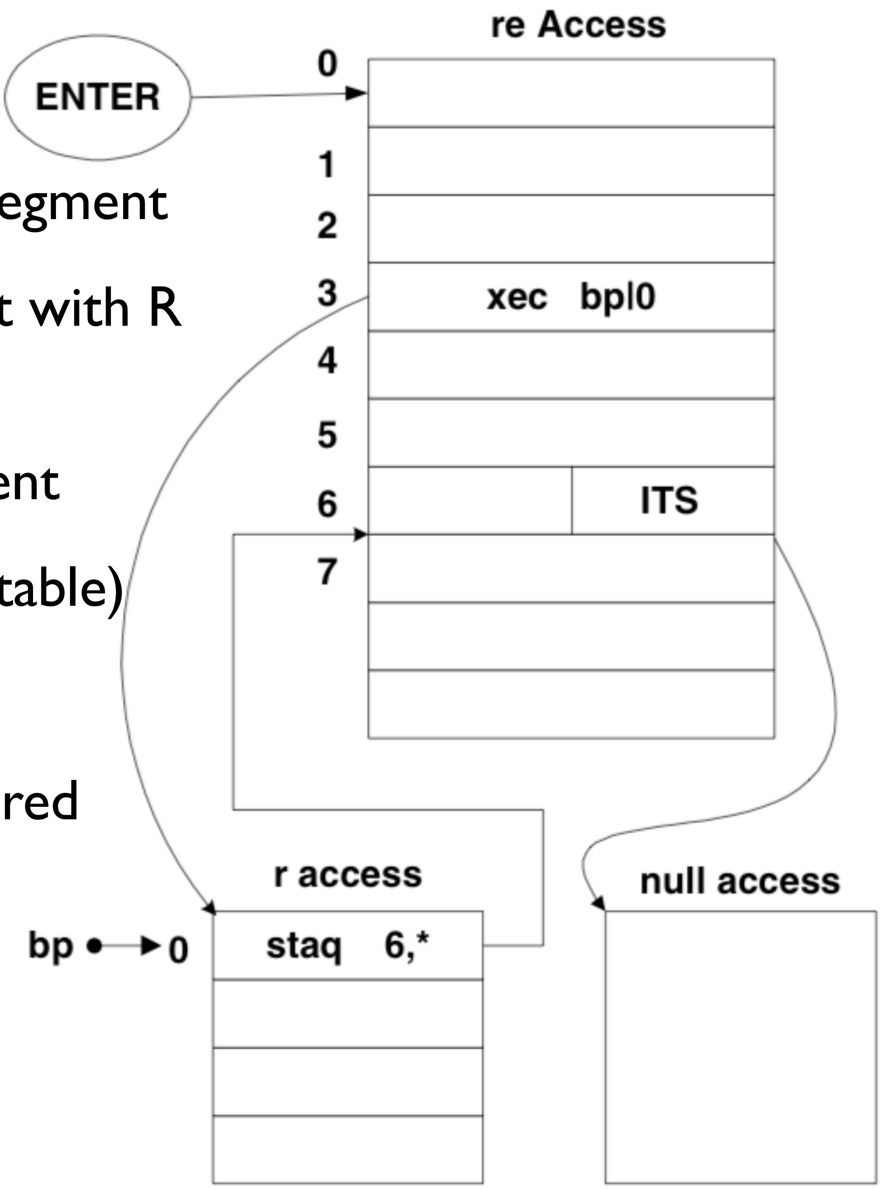
1. Hardware Vulnerability

- Hypothesis: Hardware failures violate the assumptions that underpin the security model, could violate reference monitor concept.
- Methodology:
  - Run the system for a long time
  - Each minute, invoke `subverter` to perform 1 of 22 probes to detect component failures.
- Results
  - Found one undocumented instruction discovered (not security critical?)
  - Indirect Addressing vulnerability — passing an argument that includes a reference to a second address (i.e., payload) bypasses access check on second address
    - Violates which Reference Monitor guarantee?

I. Hardware Vulnerability

<u>How to attack?</u>

1. Execute instruction with R+E access in 1st segment
2. Object instruction in word 0 of 2nd segment with R permission
3. Word for reading or writing in a third segment
4. (Third segment must already be in the page table)



<u>Result</u>: access checks for third segment are ignored

<u>Root Cause</u>: How was the error introduced?

Field modification by MIT personnel… why?

Motivate need for correctness to be verified

2. Software Vulnerability

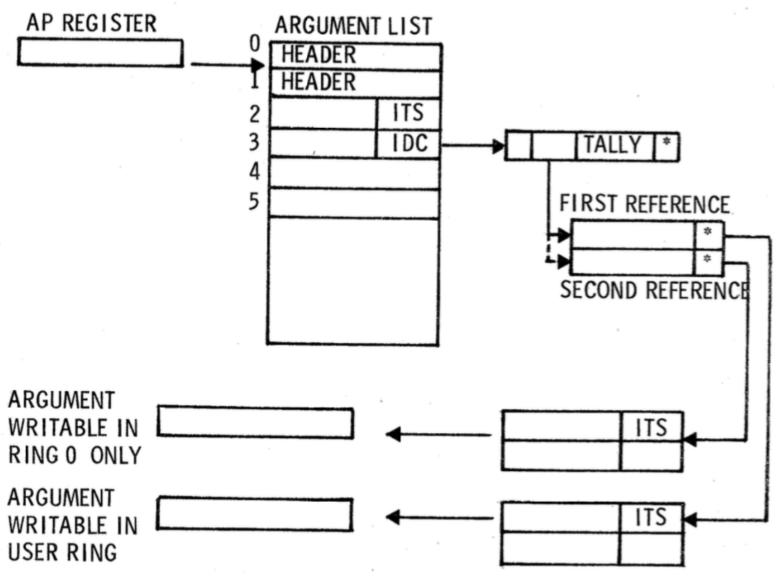
[Insufficient Argument Validation]

Origin of Vulnerability

- Early Multics did not have hardware-support for protection rings; simulated in SW instead.

"Solutions??"

- Workaround for ring-crossing — create a "gatekeeper" that validates user-supplied arguments
- What if we forget to implement a handler for a certain argument type?



Result: No validation for second referent of argument pointers that containing an IDC* modifier.

How to attack? Point second reference to an address only writable by ring 0!

The fix was ad hoc, patching IDC's but not the broader issue of input validation.

* "Increment Address, Decrement Tally, and Continue"

2. Software Vulnerability

[Master Mode Transfer]

<u>Origin of Vulnerability</u>

- Multics ran all privileged code with ring 0 permission
- This requires a trap to ring 0
- Expensive, as some privileged operations occur frequently (page faults)

<u>"Solution??"</u>

- Handle a page fault without a transition
- Justification: It has a restricted interface
- But inputs not checked!!

Be careful regarding the security impact of performance improvements

2. Software Vulnerability

[Master Mode Transfer]

<u>What did developers do wrong?</u>

- Move the master mode signaler to run in same ring as caller
- Signaler needs access to a privileged register
- Should audit this code (not done)

<u>How to attack?</u>

- Specify 0 to n-1 entry points for master mode
- Out of bounds – transfers to `mxerror`
- `mxerror` believes that a register points to signaler, but register can be modified by user (still in user's ring)

Be careful regarding the security impact of performance improvements

2. Software Vulnerability

[Unlocked Stack Base]

Origin of Vulnerability

- To reduce the complexity of Ring 0 code, designers locked the CPU register responsible for pointing to the base of the current stack (`sb`); i.e., only Master Mode code could modify `sb`.
- Simplified code because now `sb` doubles as a pointer to a valid writable memory range for fault and interrupt handlers.
- Later, language designers wanted more control over the stack (think interpretive languages like Java?)

"Solutions??": Unlock stack base, then audit Ring 0 code to remove any old assumptions about a locked `sb`

Hypothesis: ***The auditors missed a spot!***

How to attack? The `mxerror` routine contained an unaudited assumption of a locked `sb`… ultimately leads to arbitrary code execution in Ring 0.

3. Procedural Vulnerability

- Procedural Attacks
  - Tamper with the configuration of the reference validation mechanism and its dependencies
- A variety of attacks (many still used)
  - Install malicious version of system utility (e.g., `Dump`, `Patch`)
  - Forge user identities (e.g., sysadmin, security officer)
  - Modify password file
  - Hide existence of malware
  - Erase audit trails

Final Kernel Report

- Resultant system: two major problems (1974)
  - Complex: 54K LOC of code touched by hundreds of programmers
    - *Compare to today's systems… ugh.*
- Security mechanisms were ad hoc
  - Multiple mechanisms, some overlapping semantics
- Security kernel design is possible
  - Tackle later

What did Multics do right?

- No buffer overflows: choice of language made a difference here
    - Hardware support through execution bits to ensure data can’t be directly executed
    - Segmented virtual addresses
- Size: 628K for ring 0 supervisor*
    - Compare to SELinux example policy alone (1767K)
- Security auditing (though could be bypassed)
    - How to better assure the integrity of audits and collected data?
    - Motivates recent work in securing data provenance

Security Kernels

- Goals
  1. Implement a specific security policy
  2. Define a verifiable protection behavior of the system as a whole
  3. Must be shown to be faithful to the security model's design
- Recommended reading:
  - IEEE Computer, 16(7), July 1983
    (can find in IEEE Xplore)

SCOMP

Honeywell's Secure Communications Processor (SCOMP)

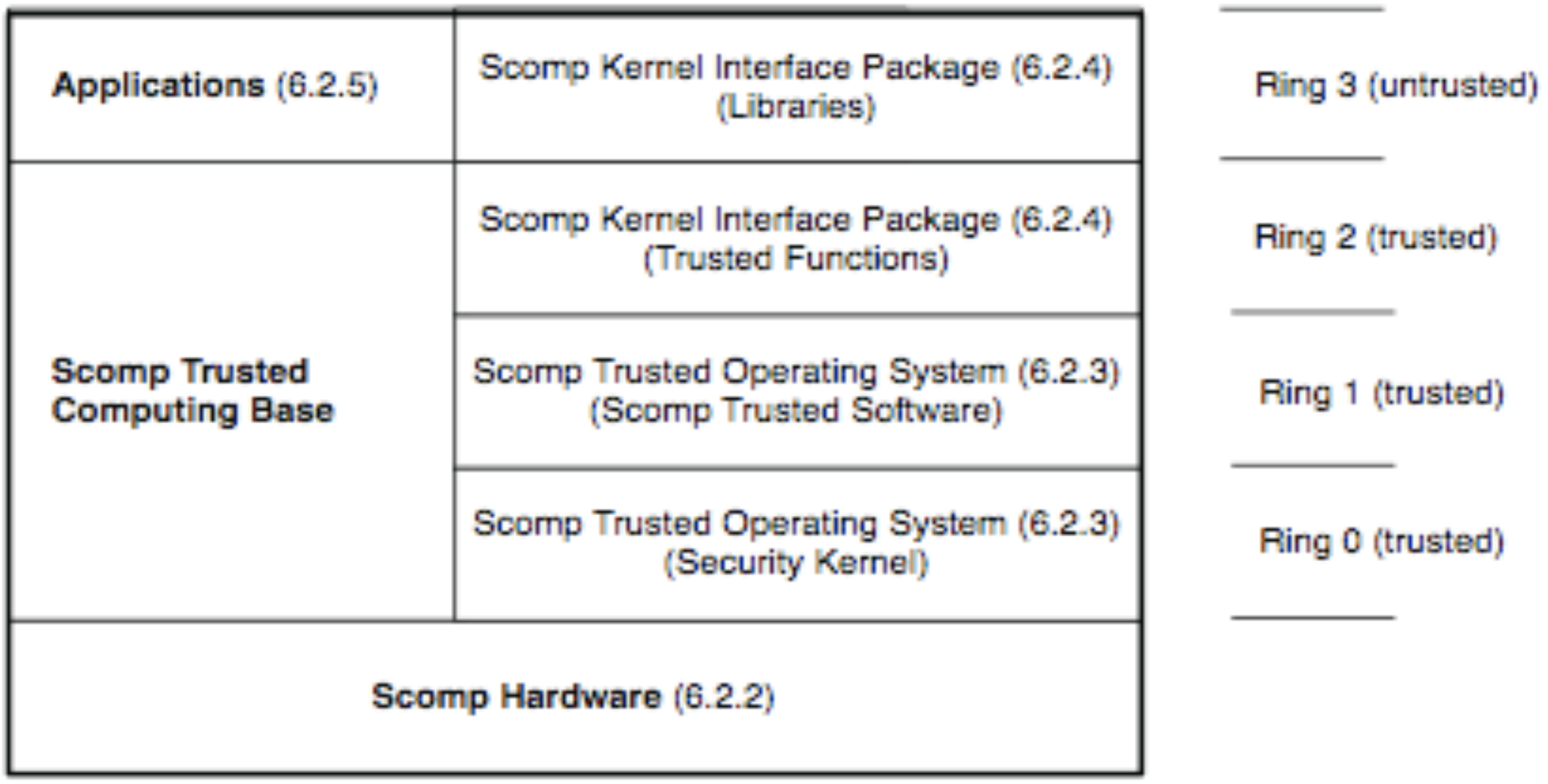

Applications (6.2.5)
Scomp Kernel Interface Package (6.2.4) (Libraries)
Ring 3 (untrusted)
Scomp Kernel Interface Package (6.2.4) (Trusted Functions)
Ring 2 (trusted)
Scomp Trusted Computing Base
Scomp Trusted Operating System (6.2.3) (Scomp Trusted Software)
Ring 1 (trusted)
Scomp Trusted Operating System (6.2.3) (Security Kernel)
Ring 0 (trusted)
Scomp Hardware (6.2.2)

Figure 6.1: The Scomp system architecture consists of hardware security mechanisms, the Scomp Trusted Operating System (STOP), and the Scomp Kernel Interface Package (SKIP). The Scomp trusted computing base consists of code in rings 0 to 2, so the SKIP libraries are not trusted.

SCOMP

Like Multics…

- Access is control via segments
  - Memory segments and I/O segments
  - Files are defined at a higher level
- Security Goals
  - Secrecy: MLS
  - Integrity: Ring brackets

SCOMP

<u>Unlike Multics…</u>

- Mediation on Segments
  - Although all access control and rings are implemented in hardware
- Formal verification
  - Verify that a formal model enforces the MLS policy
  - Trusted software outside the kernel is verified using a procedural specification
- Separate kernel from system API functions
  - In different rings (e.g., for file access)

SCOMP

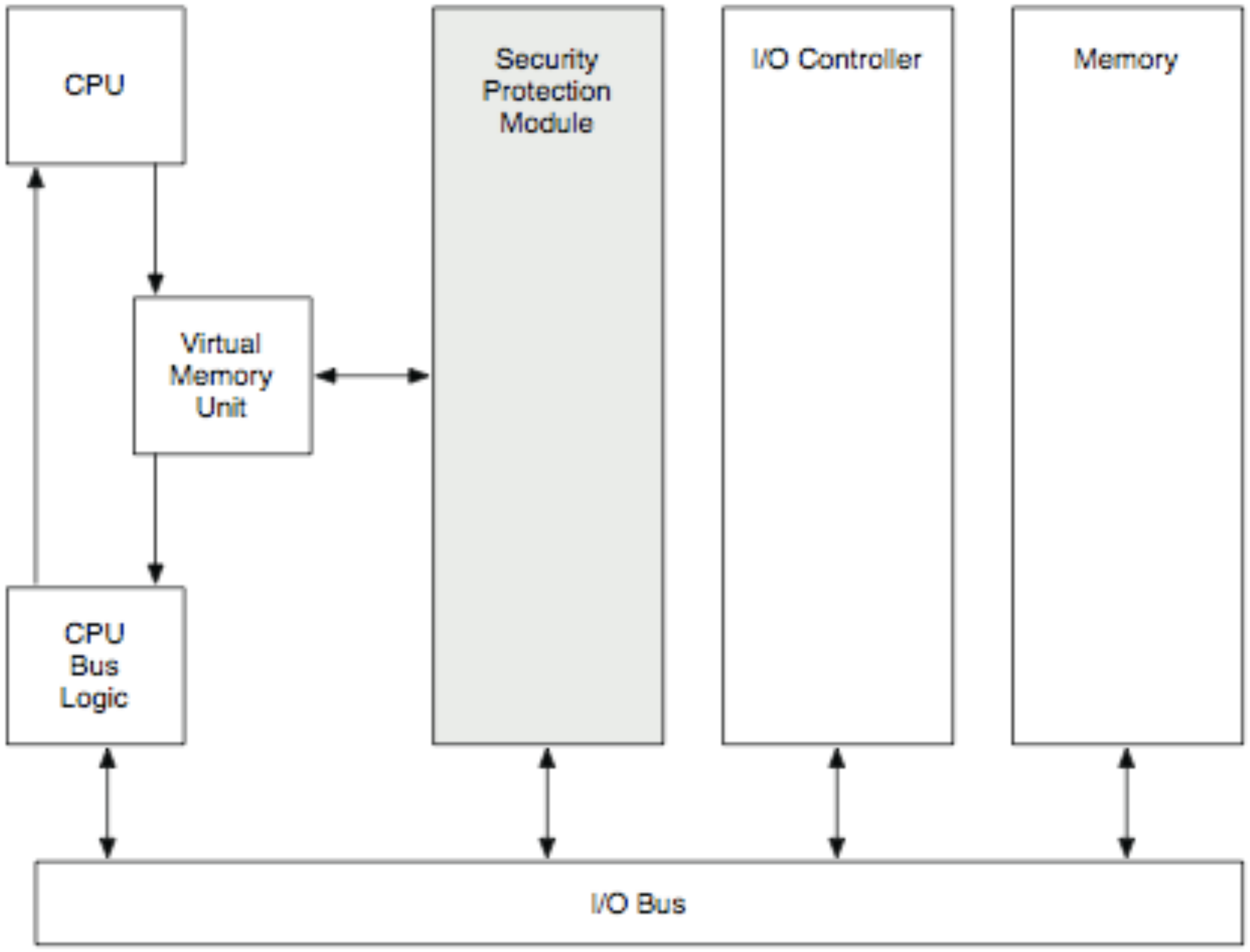


Figure 6.2: The Scomp *security protection module* (SPM) mediates all accesses to I/O controllers and memory by mediating the I/O bus. The SPM also translates virtual addresses to physical segment addresses for authorization.

SCOMP Drivers

- I/O Device Drivers in Scomp can be run in user-space
- Why can't we do that in a normal OS?
- How can we do that in Scomp?

SCOMP vs. LSM

SCOMP:

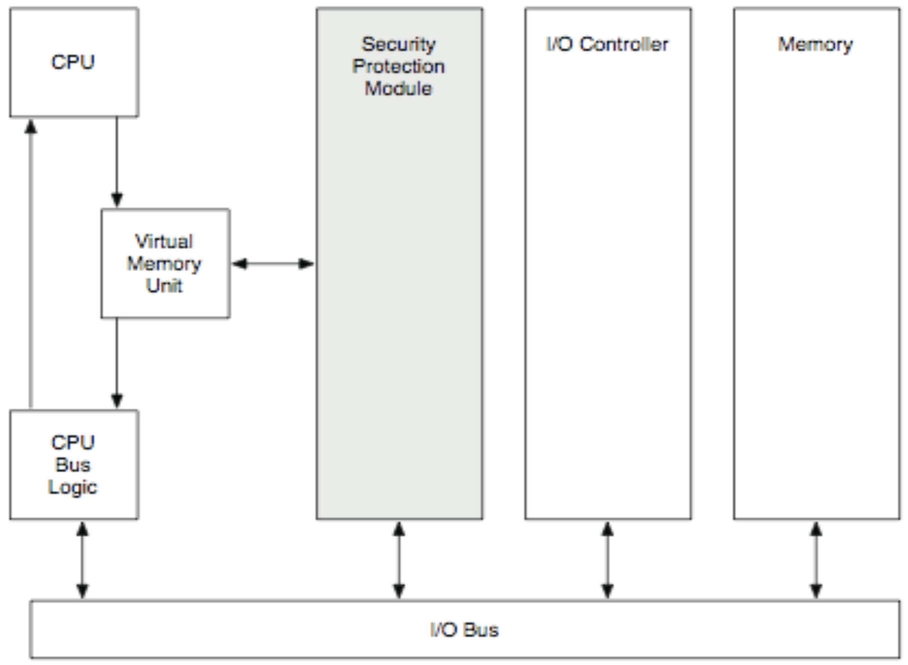
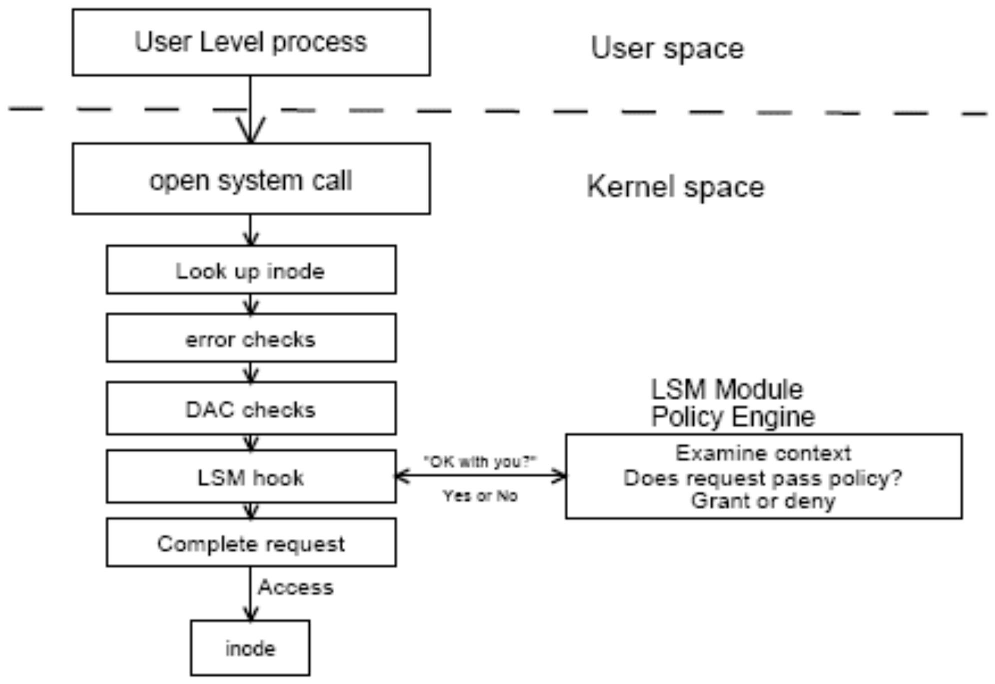


Figure 6.2: The Scomp *security protection module* (SPM) mediates all accesses to I/O controllers and memory by mediating the I/O bus. The SPM also translates virtual addresses to physical segment addresses for authorization.

Linux Security Modules:



LSM mediation occurs in software, not hardware. Affect on completeness?

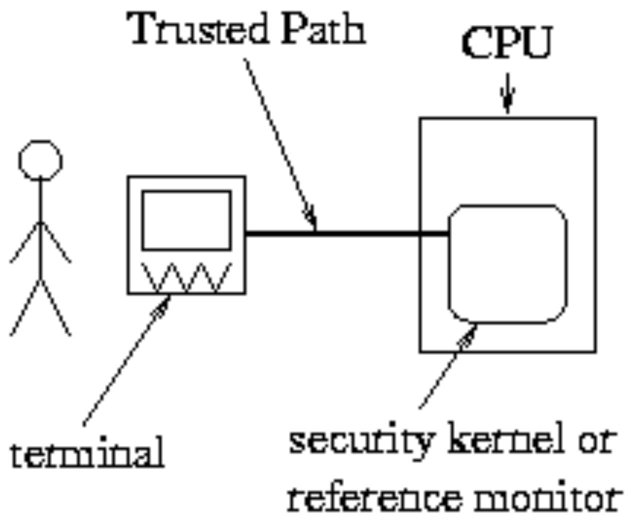
SCOMP OS

- Whole thing is called Scomp Trusted Operating Program (STOP)
  - Lives on in BEA Systems XTS-400
- Security Kernel in ring 0
  - Provides limited basic functionality: "memory management, process scheduling, interrupt management, auditing, and reference monitoring functions"
  - In 10K lines (!!!) of Pascal (!!!)
  - Ring transitions controlled by 38 gates (APIs)
    - *Can malicious user escalate privilege using gates?*

    No! The kernel doesn't even need to validate user arguments!

SCOMP Trusted Software

- Officially part of STOP
  - But runs outside ring 0
- Software trusted with system security goals
  - Like process loader
- System policy management and use
  - Such as authentication services
- 23 such processes, consisting of 11K lines of C code
  - All interaction requires a ***trusted path***
- *How does MLS inform the structure of the hierarchical file system?*



SCOMP Kernel Interface

- Like a system call interface for user processes
  - Trusted operations on user-level objects (e.g., files, processes, and I/O)
  - Still trusted not to violate MLS requirements
- Is accessible via a SKIP library
  - But that library runs in user space (ring 3)

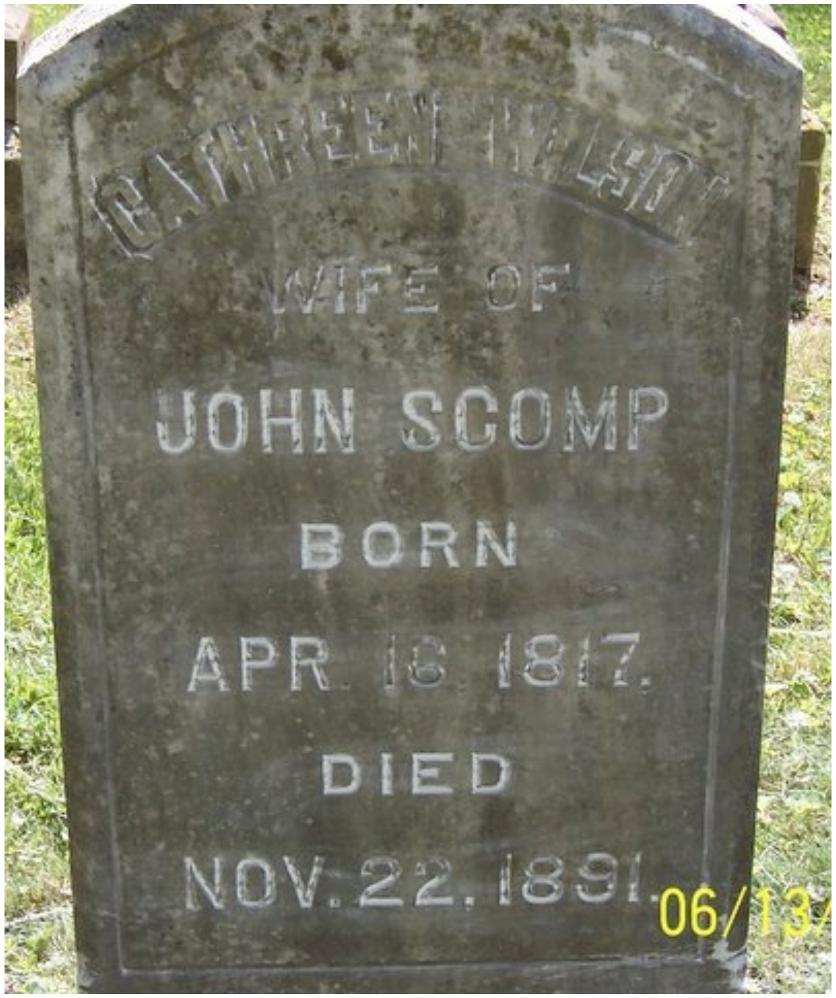
SCOMP Evaluation

- Complete Mediation: Correct?
  - In hardware
  - In Trusted programs?
- Complete Mediation: Comprehensive?
  - At segment level
  - For files?
- Complete Mediation: Verified?
  - Hardware; Trusted programs? Mail guards?

SCOMP Evaluation

- Tamperproof: Reference Monitor?
  - In hardware, in kernel, in guard
- Tamperproof: TCB?
  - TCB is well-defined in rings, and protected by gates
- Verify: Code?
  - Performed verification on implementation using semi-automated methods
  - Led to assurance criteria and approach
- Verify: Policy?
  - MLS is security goal; Integrity is more difficult

Why don’t we all use SCOMP-based systems now?

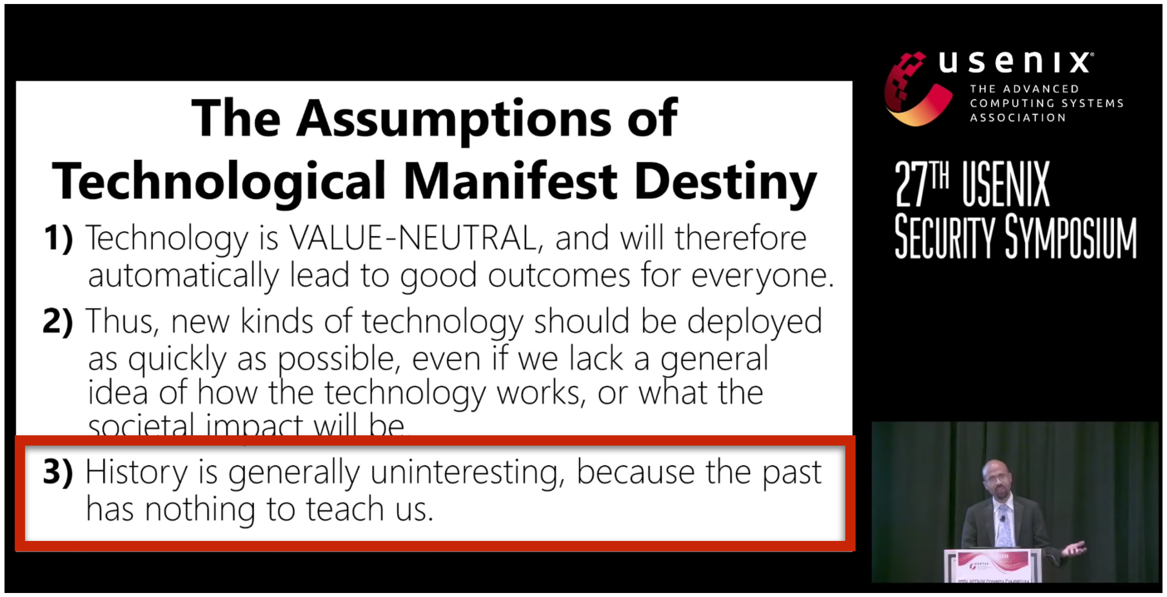


Foundations Topic: Looking Forward

- ***"Where does the quest for a security kernel pick-up after SCOMP??"***
  - e.g., GEMSOS (the SCOMP of x86 architectures),
- ***"What other primitives have been proposed for OS security?"***
  - a.k.a. ***"What DIDN'T Multics do first?"***
  - e.g., Capability systems like ICAP, Capsicum,
  - e.g., Virtual Machine Monitors like VAX VMM
  - e.g., DIFC systems like Flume, Asbestos, HiStar
- ***"Where is the security kernel today?"***
  - e.g., LSM, Subdomains, SELinux, seL4, Nested Kernels

Foundations Topic: Looking Forward

- *"Why should I go out of my way to read old esoteric papers?"*
  - Answer: Combat the evils of Technological Manifest Destiny!!



The Assumptions of Technological Manifest Destiny

1) Technology is VALUE-NEUTRAL, and will therefore automatically lead to good outcomes for everyone.

2) Thus, new kinds of technology should be deployed as quickly as possible, even if we lack a general idea of how the technology works, or what the societal impact will be.

3) History is generally uninteresting, because the past has nothing to teach us.

usenix
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION
27TH USENIX SECURITY SYMPOSIUM

Understanding classical security concepts will make your research better.

Without foundational knowledge, you'll spend your career just following shallow trends.