



Systems and Internet Infrastructure Security

Network and Security Research Center
Department of Computer Science and Engineering
Pennsylvania State University, University Park PA

Advanced Systems Security: Multics

*Trent Jaeger
Systems and Internet Infrastructure Security (SIIS) Lab
Computer Science and Engineering Department
Pennsylvania State University*

Common Knowledge

- Paraphrase
 - If people just used Multics, we would be secure
 - UNIX and Windows are insecure
- What is the basis for these statements?



Does Multics Implement

- A **Mandatory Protection System**
- Enforced by a **Reference Monitor?**



Evaluation Criteria

- **Mediation:** Does interface mediate?
- **Mediation:** On all resources?
- **Mediation:** Verifiably?
- **Tamperproof:** Is reference monitor protected?
- **Tamperproof:** Is system TCB protected?
- **Verifiable:** Is TCB code base correct?
- **Verifiable:** Does the protection system enforce the system's security goals?
- *Does Multics satisfy these?*

Multics

- Major Effort: *Multics*
 - Multiprocessing system -- developed many OS concepts
 - Including security
 - Begun in 1965
 - Development continued into the mid-70s
 - Used until 2000
 - Initial partners: MIT, Bell Labs, GE/Honeywell
- Subsequent proprietary system, *SCOMP*, became the basis for secure operating systems design

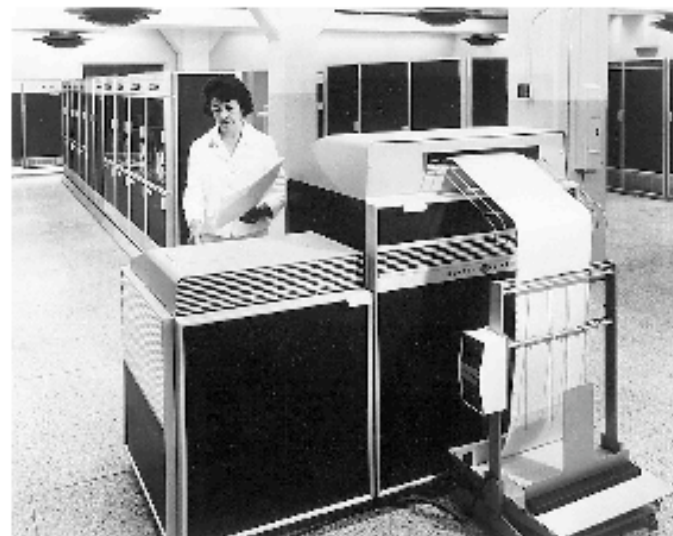


Multics Security

- What were the security goals for Multics?
 - Evolved as the system design evolved
 - First system design to consider such goals
- Secrecy
 - Prevent leakage – even if running bad code
- Integrity
 - Prevent unauthorized modification – by bad code
- Comprehensive control (enforce at OS)

Multics Security

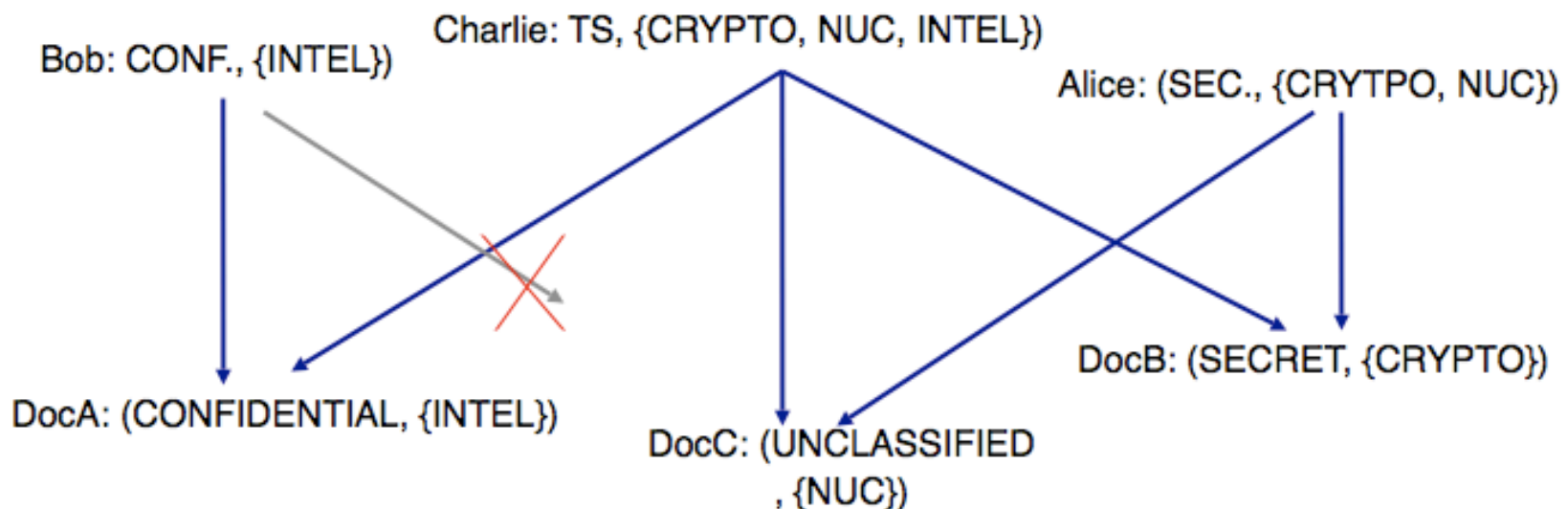
- Secrecy
 - Multilevel security
- Integrity
 - Rings of protection
- Reference Monitoring
 - Mediate segment access, ring crossing
- Resulting system is considered a high point in secure system design



- **Threat:** Trojan horse
 - A Trojan horse is a program which performs a useful function and a malicious function
 - E.g., Leaks secret data accessible to it
- Suppose a process has your secret data
 - Passwords, keys, financial info, etc.
- And executes a Trojan horse program...
- Any way you can prevent those secrets from leaking?

Multilevel Security

- Subject and object sensitivity levels
 - And categories (e.g., need to know)
- Read access (**simple security property**)
 - subject level \geq object level
 - subject categories are a superset of object categories
- Write access is opposite (***-security property**)



MLS Secrecy

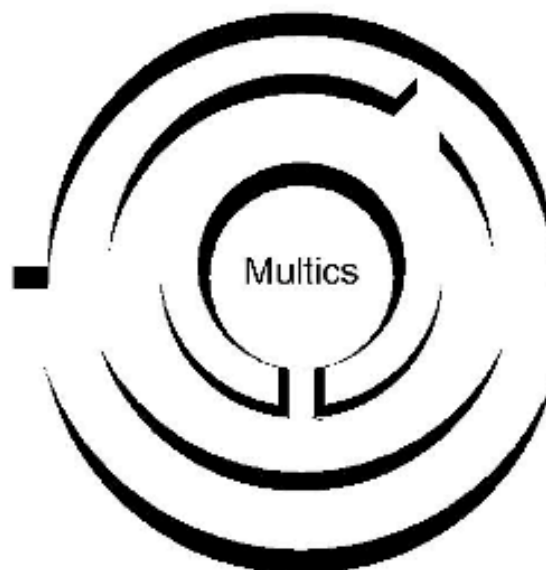
- **Threat:** Trojan horse
- Suppose a process has your secret data
 - Passwords, keys, financial info, etc.
- And executes a Trojan horse program...
- **Any way you can prevent those secrets from leaking?**
 - Yes, MLS enforcement will do that
 - How?

MLS as MPS

- Is MLS a **Mandatory Protection System?**
- **Mandatory Protection State:**
 - Level set is fixed (labels are fixed)
 - Information flows among levels are fixed
- **Labeling State:**
 - Subjects login at a level (assigned that label)
 - Objects are labeled using subject level at creation
- **Transition State:**
 - No transitions except from lower secrecy to higher

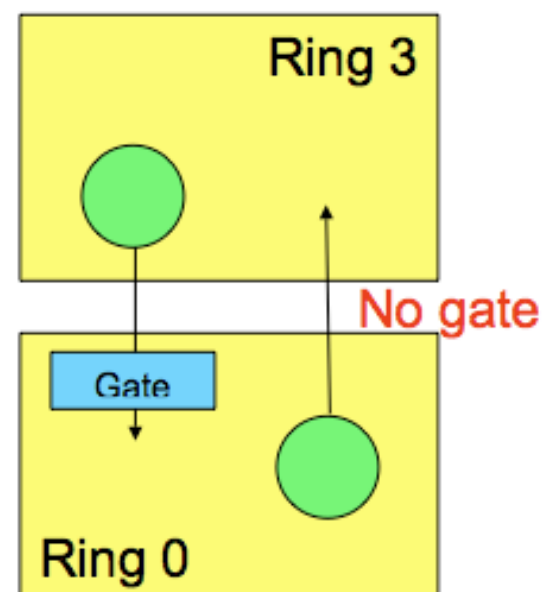
Protection Rings

- Successively less-privileged “domains”
- Example: Multics (64 rings in theory, 8 in practice)



Ring Crossing

- Program cannot call code of *higher privilege* directly
 - Gate is a special memory address where lower-privilege code can call higher
 - Enables OS to control where applications call it (system calls)



Ring Brackets

- Kernel resides in ring 0
- Process runs in a ring r
 - Access based on current ring
- Process accesses data (segment)
 - Each data segment has an *access bracket*: $(a1, a2)$
 - $a1 \leq a2$
 - Describes read and write access to segment
 - r is the current ring
 - $r \leq a1$: access permitted
 - $a1 < r \leq a2$: r and x permitted; w denied
 - $a2 < r$: all access denied

Procedure Invocation Brackets

- Also different procedure segments
 - with *call brackets*: (c1, c2)
 - $c1 \leq c2$
 - and access brackets (a1, a2)
 - Rights to execute code in a new procedure segment
 - $r < a1$: access permitted with ring-crossing fault
 - $a1 \leq r \leq a2 = c1$: access permitted and no fault
 - $a2 < r \leq c2$: access permitted through a valid gate
 - $c2 < r$: access denied
- What's it mean?
 - case 1: ring-crossing fault changes procedure's ring
 - increases from r to a1
 - case 2: keep same ring number
 - case 3: gate checks args, decreases ring number

Brackets Examples

- Authorized or not?
- Process in ring 3 accesses data segment
 - access bracket: (2, 4)
 - What operations can be performed?
- Process in ring 5 accesses same data segment
 - What operations can be performed?
- Process in ring 5 accesses procedure segment
 - access bracket (2, 4) and call bracket (4, 6)
 - Can call be made? How do we determine the new ring? Can new procedure segment access the data segment above?

Brackets as MPS

- Are brackets a **Mandatory Protection System?**
- **Mandatory Protection State:**
 - Rings are fixed in a hierarchy
 - **Protection state can be modified by owner**
- **Labeling State:**
 - Ring determined at login
 - **Owner can change object's ring**
- **Transition State:**
 - Thru call brackets (guarded by gates)

Multics Reference Monitor

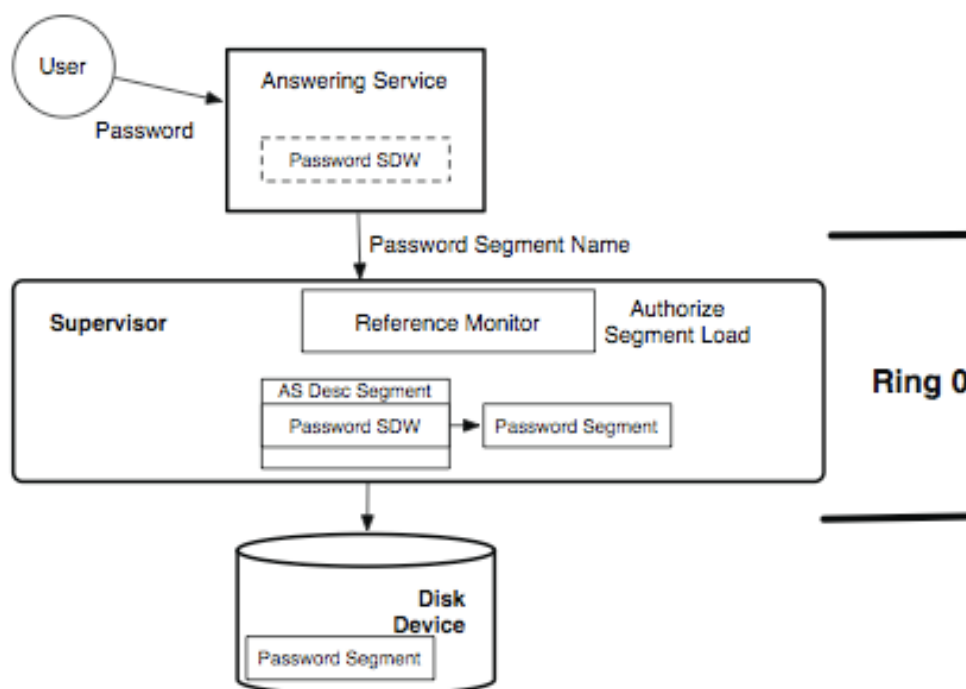


Figure 3.2: The Multics login process. The user's password is submitted to the Multics *answering service* which must check the password against the entries in the *password segment*. The Multics *supervisor* in the privileged *protection ring 0* authorizes access to this segment and adds a SDW for it to the answering service's descriptor segment. The answering service cannot modify its own descriptor segment.

SDW Format

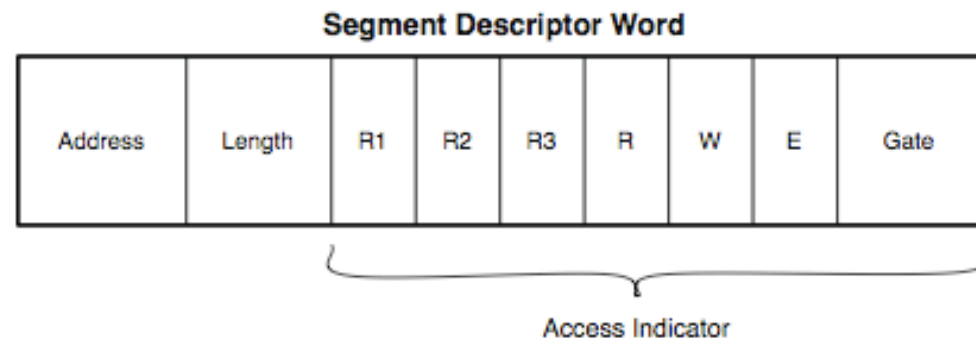


Figure 3.3: Structure of the Multics *segment descriptor word* (SDW): in addition to the segment's address and length, the SDW contains access indicators including *ring brackets* (i.e., *R1*, *R2*, *R3*), the process's ACL for the segment (i.e., the *rwe* bits), and the *number of gates* for the segment.

- Process uses SDW to access a segment
 - Directory stores a mapping between segments and secrecy level
 - Each segment has a ring bracket specification
 - Copied into SDW
 - Each segment has an ACL
 - Authorized ops in RWE bits

SDW Examples

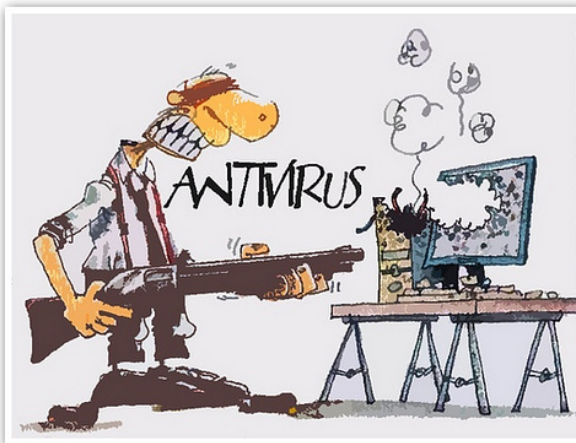
- Read authorized or not?
- Secrecy
 - Clearance of process - secret
 - Access class of segment - confidential
- Brackets
 - Process in ring 2
 - Access bracket (2-3); Call bracket (4-5)
- Access control list
 - RWE

Multics Reference Monitor

- Mediation
 - Security-sensitive operations on [segments](#)
 - All objects are accessed via a named hierarchy of segments
 - Predates file system hierarchies; other objects?
- Tamperproofing
 - Reference monitor is part of the kernel ring
 - Minimize dependency on software outside kernel ([call brackets](#))
- Verifiability
 - Lots of code (well, 54K SLOC, but [too much to verify formally](#))
 - MLS for secrecy and rings/brackets for integrity ([not mandatory](#))

So How Secure?

- So, Multics fails to meet **mandatory protection state** and **reference monitor** guarantees – is that so bad?
 - ▶ Still possible to configure integrity (if TCB cannot be compromised)
 - ▶ There's a lot of code and complex concepts, but we can handle it
 - ▶ Right?



Vulnerability Analysis

- Background
 - ▶ 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 “secureable” (1.3.3)
 - ▶ Based on security descriptor mediation
 - ▶ Ring protection

Vulnerability Analysis

- Criteria details
 - Is reference monitor practical for Multics?
 - Identify necessary security enhancements
 - Determine scope of a certification effort
- Logistics
 - At MIT (developers/users) + At Rome ADC (Air Force users)
 - Honeywell 645 running a Multics system (old HW)

Hardware Vulnerability

- Run the system for a long time
 - Didn't crash, but
 - Found one undocumented instruction and one vulnerability
- Indirect Addressing
 - Address provided references the actual address to use
 - Mechanism only checked the first address
- Result
 - Bypass access checking (fails complete mediation)

Hardware Vulnerability

- How to attack?
 - Execute instruction with RX access in first segment
 - Object instruction in word 0 of second segment with R permission
 - Word for reading or writing in a third segment
 - Third segment must already be in the page table
- Access checks for third segment are ignored
 - Do whatever to contents on this third segment
- Motivate need for correctness to be verified

Other Design Details

- **Master Mode**
 - ▶ Procedures used in ring 0 to run privileged functions
 - What are these analogous too in modern systems?
 - ▶ “Pseudo-operation code” at location 0 in ring 0
 - Start at a well-known location (gate)
 - ▶ Test the entry point for validity
 - Only run known function from known locations
- Avoid trying to run privileged code that may be impacted by users

Software Vulnerability

- Master mode vulnerability
 - Run privileged code with ring 0 perms
 - Requires a trap to ring 0
 - Expensive as some privileged operations occur frequently (page faults)
- Change: Handle a page fault without a transition
 - Justification: It has a restricted interface
 - But inputs not checked
- Bingo – Be careful regarding the security impact of performance improvements

Software Vulnerability

- What developers did wrong?
 - Move the master mode **signaller** to run in same ring as caller
 - Signaler stored in a privileged register
- How to use?
 - 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)

Final Kernel Report

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

Evaluation Criteria

- **Mediation:** Does interface mediate correctly?
 - ▶ Mediates on object references
 - ▶ Indirect addressing (multiple objects references)
- **Mediation:** On all resources?
 - ▶ All objects are segments
 - ▶ What would happen if network was introduced?
- **Mediation:** Verifiably?
 - ▶ Uh, working on it
 - ▶ Some use complex formats, so such verification is required

Evaluation Criteria

- **Tamperproof:** Is reference monitor protected?
 - In supervisor, with trusted code
 - Access via gates and master mode in controlled way (mostly)
- **Tamperproof:** Is system TCB protected?
 - Managed by brackets
 - Can modify brackets; moved master mode code out of ring 0
- **Verifiable:** Is TCB code base correct?
 - Trying to verify
 - Didn't verify (did later, but not fully)
- **Verifiable:** Does the protection system enforce the system's security goals?
 - Not an MPS

Take Away

- Multics originated the development of a “secure operating system”
 - Real attempts were made to achieve **reference monitor guarantees** and provide a **mandatory protection system** (e.g., MLS)
- However, it is not easy to satisfy reference monitor guarantees, even when you try
 - Especially, if your system maintainers are not trying
- And if you are not trying to satisfy RM guarantees
 - You won't have anything close (UNIX and Windows)