Systems and Internet Infrastructure Security

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

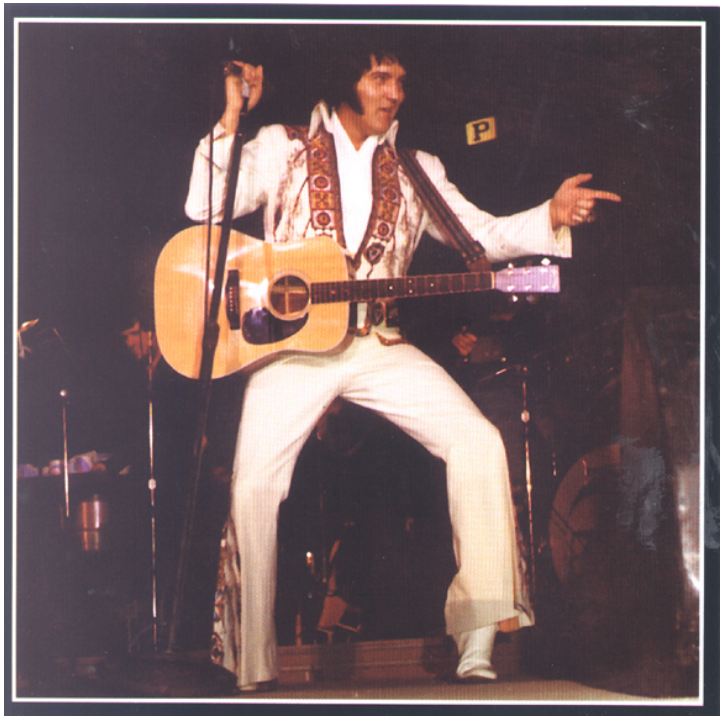
Advanced Systems Security: Security Kernels

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

Multics circa 1976

PENNSTATE 1855

- Final research report
  - Slower than desired
  - Bigger than desired
  - Expensive ($7M)
  - Low market share
  - UNIX winning mindshare
- Next generation systems?



Two Directions

- Focus on Generality and Performance
  - Limited security
  - Focus: UNIX
  - Put us in our current state
- Focus on "verifiable" security
  - Security kernels
  - Lots of systems
    - *KSOS, PSOS, Secure LAN, Secure Ada Target, various guard systems*

MITRE Project

- Started in 1974
  - OS in 20 subroutines
    - Less than 1000 lines of code
  - System to manage physical resources
- What are the advantages of such an approach?

Security Kernel

PENNSTATE 1855

- Goals
- *(1) Implement a specific security policy*
- *(2) Define a verifiable protection behavior of the system as a whole (reference monitor)*
- *(3) Must be shown to be faithful to the security model's design (reference monitor enforces policy)*
- Recommend a special issue
  - IEEE Computer, 16(7), July 1983

Verification

PENNSTATE 1855

- Became the focus of the approach
  - Verify that the implementation is faithful to the model
  - Which supports a specific security policy
- What are the formal limits of verification?
- What is it going to take in this case?

Scomp

- Secure Communications Processor (Scomp)

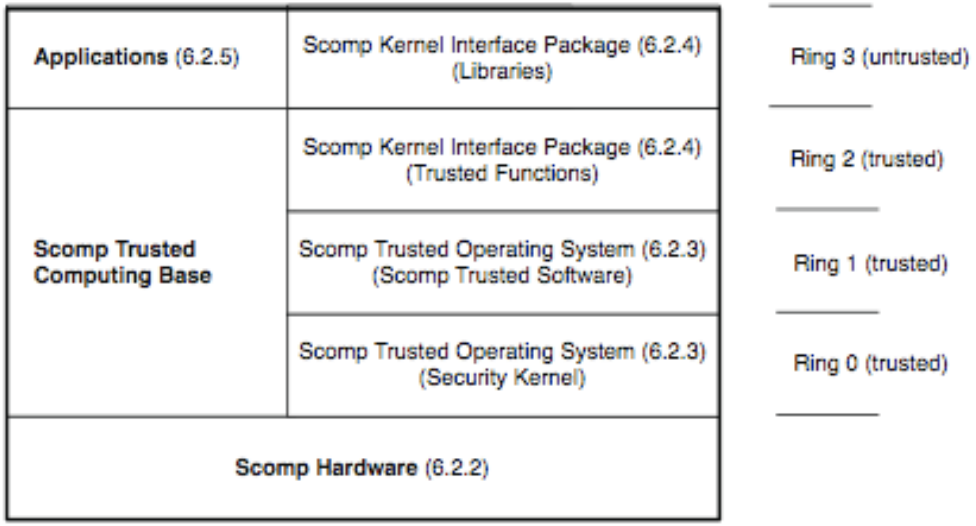


Figure 6.1: The Scomp system architecture consists of hardware security mechanisms, the Scomp Trusted Operating System (STOP), and the Scomp Kernel Interface Package (SKIP). The Scomp trusted computing base consists of code in rings 0 to 2, so the SKIP libraries are not trusted.

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

- Unlike Multics
- Mediation on Segments
  - 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 Hardware

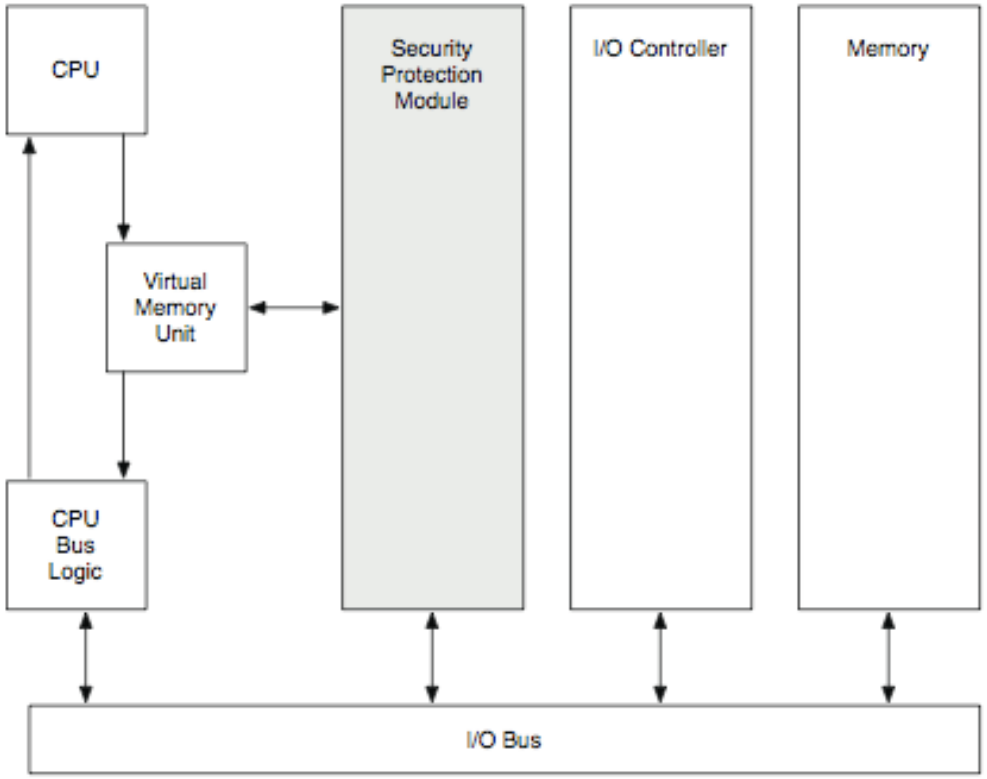


Figure 6.2: The Scomp *security protection module* (SPM) mediates all accesses to I/O controllers and memory by mediating the I/O bus. The SPM also translates virtual addresses to physical segment addresses for authorization.

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 OS

- Whole thing is called Scomp Trusted Operating Program (STOP)
  - Lives on in BEA Systems XTS-400
- Security Kernel in ring 0
  - Provides "memory management, process scheduling, interrupt management, auditing, and reference monitoring functions"
  - In 10K lines of Pascal
  - Ring transitions controlled by 38 gates (APIs)

Trusted Software

- Officially part of STOP
  - But runs outside ring 0
- Software trusted to 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

Scomp Kernel Interface

PENNSTATE 1855

- 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)
  - Like libc and POSIX API…

Scomp Applications

PENNSTATE 1855

- May also be MLS-Trusted Applications
- Mail Guard
  - Makes sure that secrets are not leaked in communications to less secret subjects
  - Mail guard obtains labeled communications
    - Has ad hoc filters to prevent leakage
- Why is Scomp appropriate to support such an application?

Scomp Evaluation

- **Complete Mediation**: Correct?
  - In hardware
  - In Trusted programs? In Mail guards?
- **Complete Mediation**: Comprehensive?
  - At segment level
  - For files? For mail data? For DMA operations?
- **Complete Mediation**: Verified?
  - Hardware? Kernel? 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

Scomp Challenges

PENNSTATE 1855

- Why don't we all use Scomp-based systems now?

GEMSOS

- Similar system goals to Scomp
- Built for the 'new' x86 processor

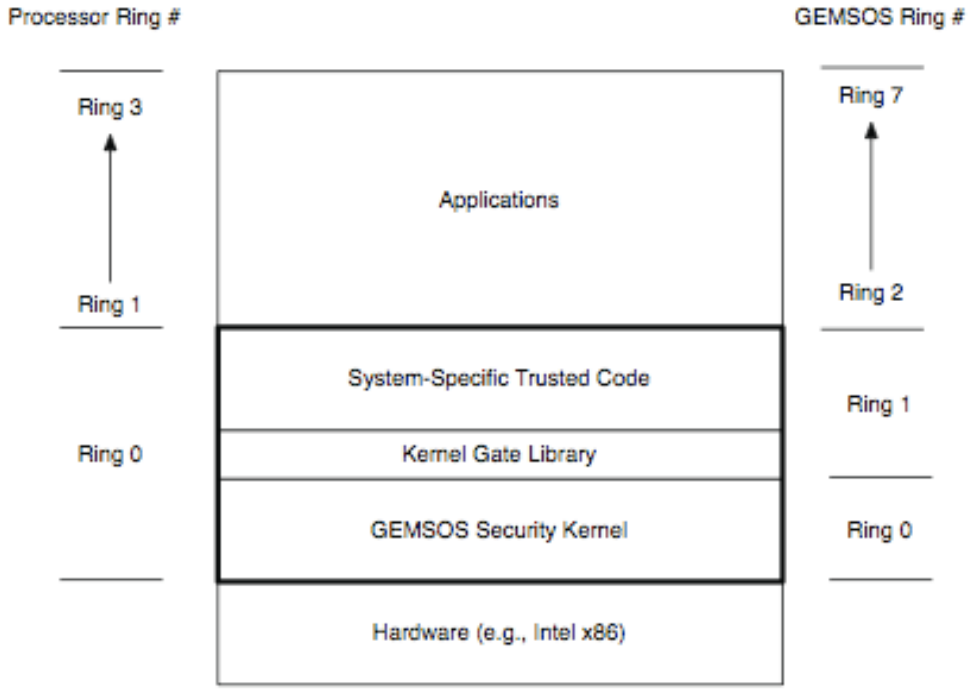


Figure 6.3: GEMSOS consists of a security kernel, gate library, and a layer of trusted software that is dependent on the deployed system. GEMSOS uses a software-based ring mechanism to simulate 8 protection rings.

GEMSOS

- Also, is still around
  - Aesec corporation
- Fine-grained kernel design
- Eventually, UNIX (POSIX) emulation
- File system is inside the security kernel
  - Kernel and trusted software depends on data layout in files

Applications	
Gate Layer	↑
Process Manager (PM)	Process
Upper Device Manager (UDM)	Local
Segment Manager (SM)	↓
Upper Traffic Controller (UTC)	↑
Memory Manager (MM)	|
Inner Device Manager (IDM)	|
Secondary Storage Manager (SSM)	|
Non-Discretionary Security Manager (NDSM)	Kernel
Kernel Device Layer (KDL)	Global
Inner Traffic Controller (ITC)	|
Core Manager (CM)	|
Intersegment Linkage Layer (SG)	|
System Library (SL)	↓
Hardware	

Figure 6.4: GEMSOS Security Kernel Layers

Driver Isolation

PENNSTATE 1855

- A big claim in Scomp was the ability to run drivers securely in user space
  - By mediating access between the CPU and I/O Bus
  - But, later technology resulted in incomplete mediation
- The introduction of IOMMUs may enable effective driver isolation
  - How?
  - How do we use it effectively?
- Those are the topics of Herder et al

Why are drivers a problem?

PENNSTATE 1855

- System errors cause the biggest problems
  - Unplanned downtime is mainly due to faulty system software
- Many system errors are caused by drivers
  - Responsible for "majority of OS crashes"
  - 2/3 of the code base is due to "extensions", but they have an error rate 3-7 times higher than other code
  - 65-83% of all crashes in Windows XP due to drivers
- How do we reduce such problems?

Can we fix drivers?

PENNSTATE 1855

- There are many drivers and they change and new ones emerge rapidly
- They are a large portion of the kernel code
  - 3M lines compromising nearly 60%
- We know that we must isolate drivers
  - But how?
- News: IOMMU

Driver isolation previously

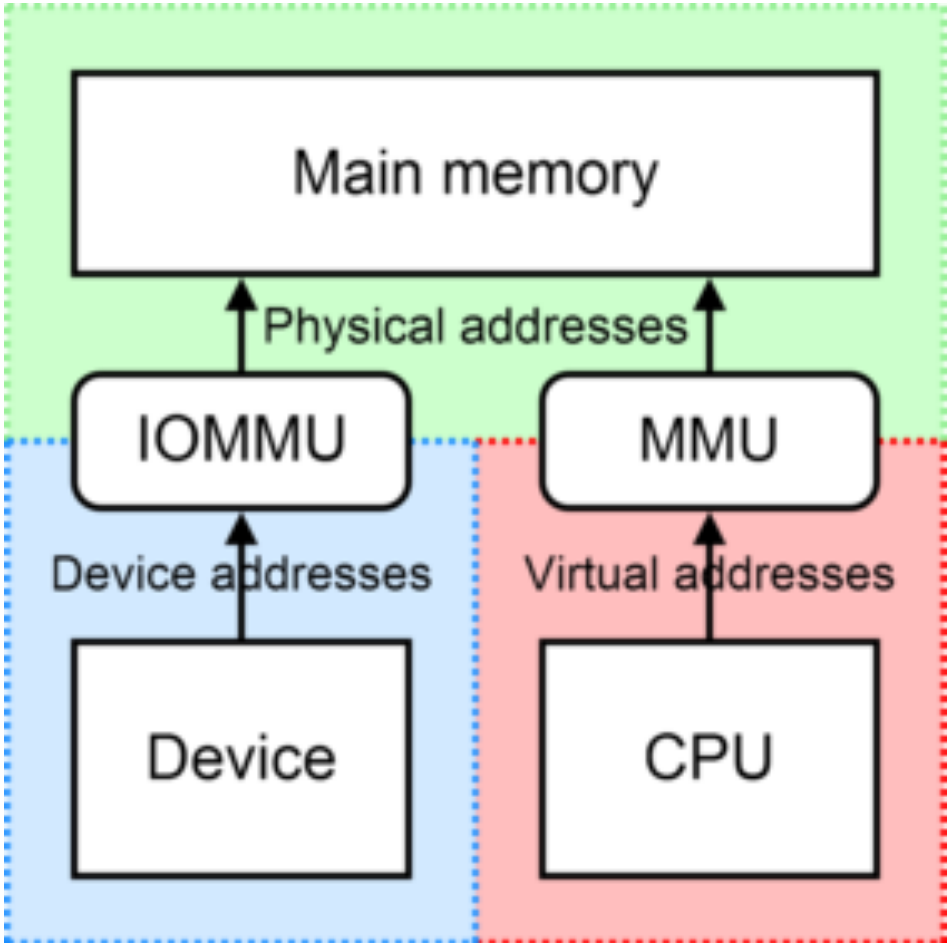
PENNSTATE 1855

- Build on: trusted kernel and MMU
- Isolate drivers and wrap their invocations
  - MMU enables drivers operations to be isolated from the kernel
  - But drivers can cause devices to write to privileged memory (how?)
  - So, the trusted kernel needs to check requests sent to the device (for many drivers and devices)
- Also, use programming language, virtual machine, etc.
- My experience SawMill Linux Multiserver [11]

MMU

- What exactly does an MMU do?
  - Maps virtual to physical addresses using the process's page tables
- Placing a driver in a limited memory context restricts which pages it can access
- We get a boundary, but there are many holes thru
  - But the driver has to access the device (DMA)
  - And it also performs operations that the OS depends upon (OS services)

IOMMU


Main memory
Physical addresses
IOMMU
MMU
Device addresses
Virtual addresses
Device
CPU

Limitations

- In addition to memory protection…
- Must deal with multiple devices sharing an interrupt line
  - One may block the line
- Sharing of the PCI bus
  - Conflicts between devices
- Performance overhead of isolation
  - 10-25% in macrobenchark
  - A lot more on microbenchmarks

Overview of Approach

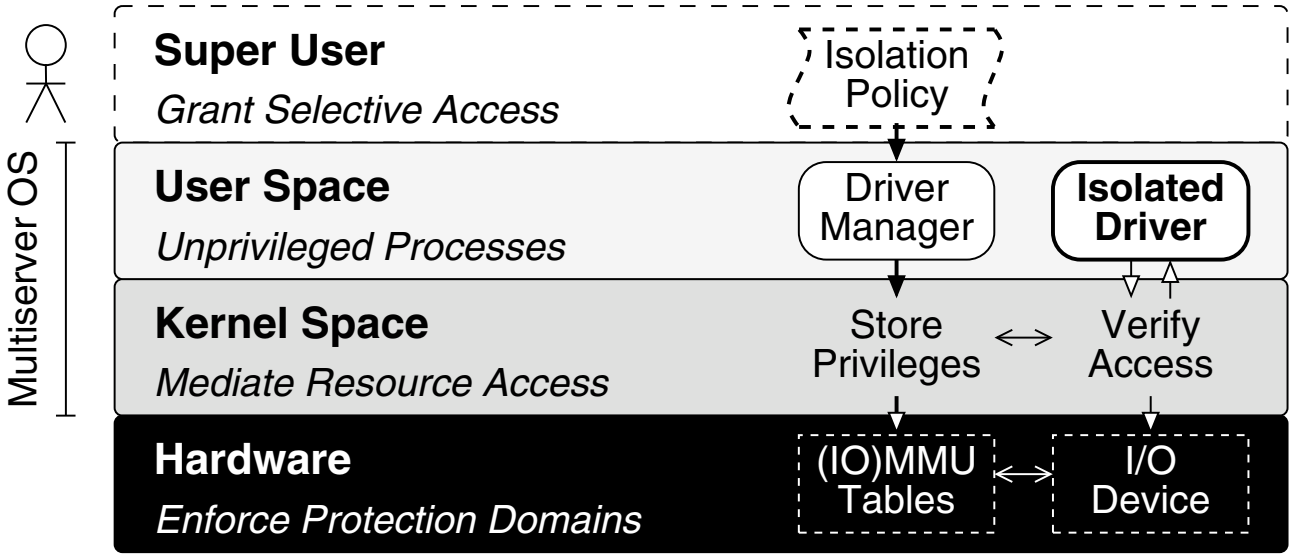


Figure 2: MINIX 3 isolates drivers in unprivileged processes.

Classification of Privilege Mgmt

Privileges	Isolation Techniques
(Class I) CPU Usage	**See Sec. 4.2.1**
+ *Privileged instructions*	→ *User-mode processes*
+ *CPU time*	→ *Feedback-queue scheduler*
(Class II) Memory access	**See Sec. 4.2.2**
+ *Memory references*	→ *Address-space separation*
+ *Copying and sharing*	→ *Run-time memory granting*
+ *Direct memory access*	→ *IOMMU protection*
(Class III) Device I/O	**See Sec. 4.2.3**
+ *Device access*	→ *Per-driver I/O policy*
+ *Interrupt handling*	→ *User-level IRQ handling*
(Class IV) System services	**See Sec. 4.2.4**
+ *Low-level IPC*	→ *Per-driver IPC policy*
+ *OS services*	→ *Per-driver call policy*

Figure 3: Classification of privileged operations needed by low-level drivers and summary of MINIX 3's defense mechanisms.

Complete Mediation

- Privilege Instructions
  - Driver in user-space
- CPU time
  - Scheduling
- Memory References
  - Address space isolation
- DMA
  - IOMMU protection
- IRQ
  - User-level IRQ handling

- Sharing with driver
  - Grant mechanism
- Access to services
  - Driver syscall policy
- Driver access
  - Driver I/O policy
- IPC
  - Driver IPC policy

Sharing with Drivers

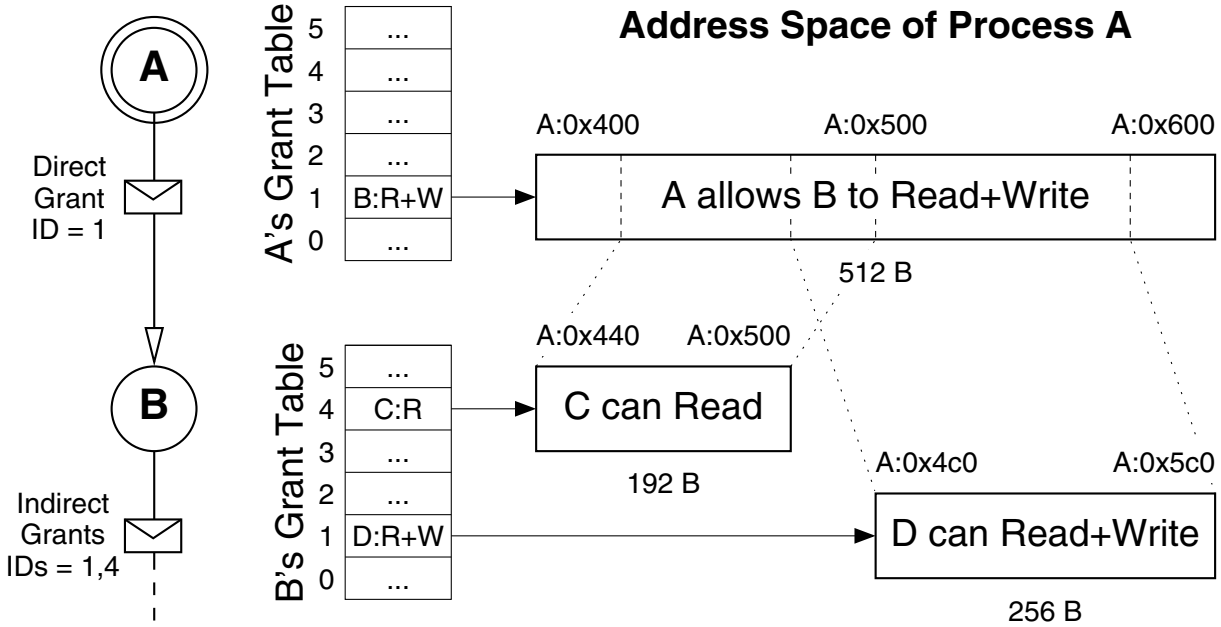


Figure 4: Hierarchical structure of memory grants. Process A directly grants B access to a part of its memory; C can access subparts of A's memory through indirect grants created by B.

Other Policies

PENNSTATE 1855

- Specify a declarative, least privilege policy for each driver
- Driver I/O
  - PCI: device or class
  - ISA: I/O ports and IRQs
- IPC
  - Who can the device send IPC to?
- OS Services
  - What kernel interfaces are available? OS space services?

Mandatory Protection System

- Does this approach implement a mandatory protection system?
- Protection State: devices and interfaces
  - Fig 5 policy – mandatory?
  - Distributed among several components
  - Sharing – setup under discretion, but?
- Labeling State
- Transition State

Security Analysis

- Complete Mediation?
  - Who does mediation?
  - Are they all reference monitors?
- Tamperproofing
  - Reference monitors and who they depend upon
- Verification
  - Code and Policy
- Does this approach satisfy reference monitor concept?

Take Away

- Security kernel design approach was designed to address security shortcomings of Multics
  - In particularly, size and complexity
- Security kernel design approach
  - Documented in a book by Morrie Gasser
  - Led to the assurance approach in the Orange Book
- When people speak of how to build "secure OSes" they probably mean these systems
- So, we aren't we building systems this way?
- What ideas/approaches can we take into current systems?