Systems and Internet Infrastructure Security

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

CSE543 - Introduction to Computer and Network Security
Module: Operating System Security

Professor Trent Jaeger

OS Security

- So, you have built an operating system that enables user-space processes to access hardware resources
  - Thru various abstractions: files, pages, devices, etc.
- Now, you want your operating system to enforce security requirements for your application processes
  - What do you do?

OS Security

- We learned about a few things that will help you
- Your OS must implement a
  - (Mandatory) Protection system
- That can enforce a
  - MAC policy
- How do we implement such an OS mechanism?
  - Multics
  - Linux Security Modules

Access Policy Enforcement

- A protection system uses a *reference validation mechanism* to produce and evaluate authorization queries
  - Interface: Mediate security-sensitive operations by building authorization queries to evaluate
  - Module: Determine relevant protection state entry (ACLs, capabilities) to evaluate authorization query
  - Manage: Install protection state entries and reason about labeling and transition states

- How do we know whether a reference validation mechanism is correct?

Security-Sensitive Operations

- Broadly, operations that enable interaction among processes that violate secrecy, integrity, availability
- Which of these are security-sensitive? Why?
  - Read a file (*read*)
  - Get the process id of a process (*getpid*)
  - Read file metadata (*stat*)
  - Fork a child process (*fork*)
  - Get the metadata of a file you have already opened? (*fstat*)
  - Modify the data segment size? (*brk*)
- Require protection for all of CIA?

Reference Monitor

PENNSTATE 1855

- Defines a set of requirements on reference validation mechanisms
  - To enforce access control policies correctly
- Complete mediation
  - The *reference validation mechanism* must always be invoked (before executing security-sensitive operations)
- Tamperproof
  - The *reference validation mechanism* must be tamperproof
- Verifiable
  - The *reference validation mechanism* must be small enough to be subject to analysis and tests, the completeness of which can be assured

Multiprocessor Systems

- Major Effort: *Multics*
  - Multiprocessing system -- developed many OS concepts
    - Including security
  - Begun in 1965
    - Research continued into the mid-70s
  - Used until 2000
  - Initial partners: MIT, Bell Labs, GE (replaced by Honeywell)
  - *Other innovations*: hierarchical filesystems, dynamic linking

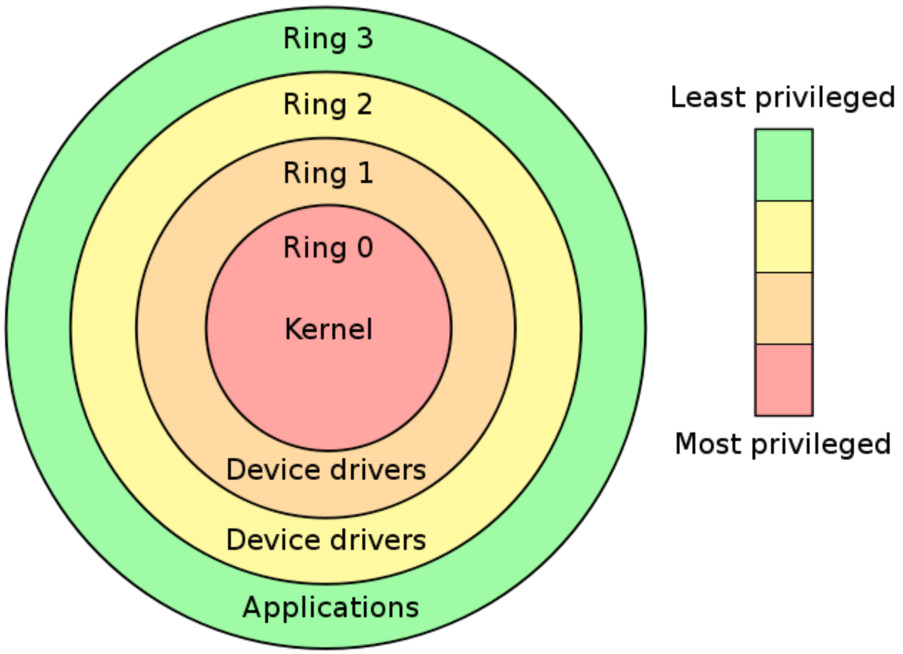
- Subsequent proprietary system, *SCOMP*, became the basis for secure operating systems design (XTS-400)

Multics Goals

- Secrecy
  - Multilevel security
- Integrity
  - Rings of protection
- Resulting system is considered a high point in secure systems design

Protection Rings

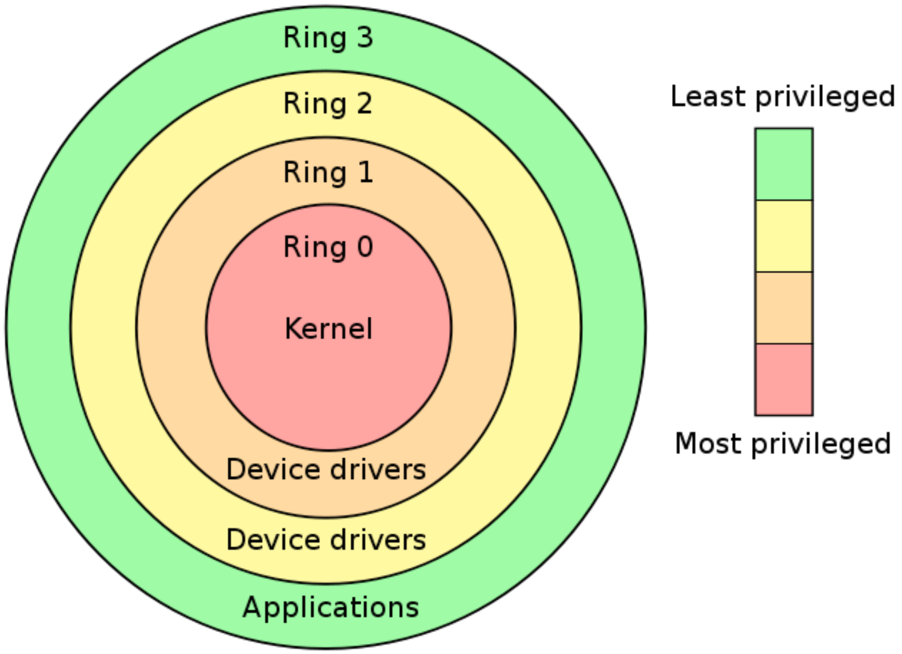
- Successively less-privileged "domains"
- Modern CPUs support 4 rings
    - Use 2 mainly: Kernel and user
- Intel x86 rings
    - Ring 0 has kernel
    - Ring 3 has application code



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

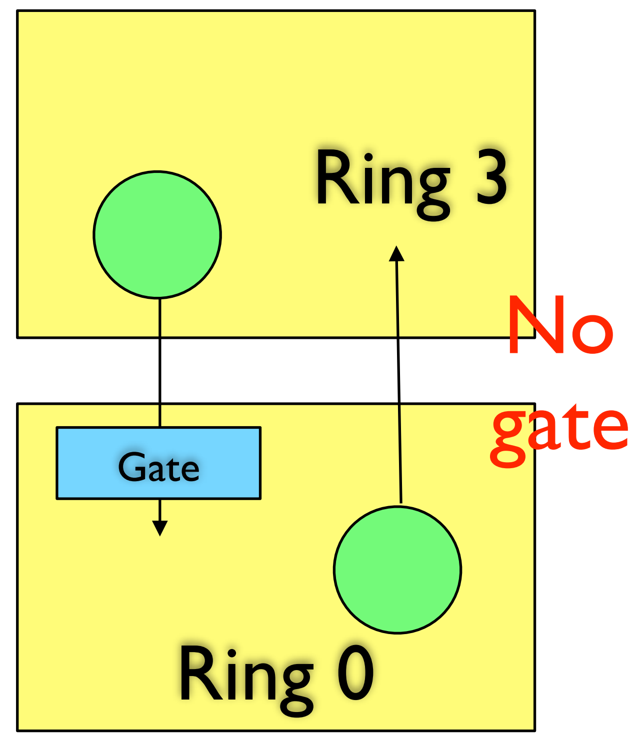
What Are Protection Rings?

- Coarse-grained, Hardware Protection Mechanism
- Boundary between Levels of Authority
  - Most privileged -- ring 0
  - Monotonically less privileged above
- Fundamental Purpose
  - Protect system integrity
    - Protect kernel from services
    - Protect services from apps
    - So on...



Protection Ring Rules

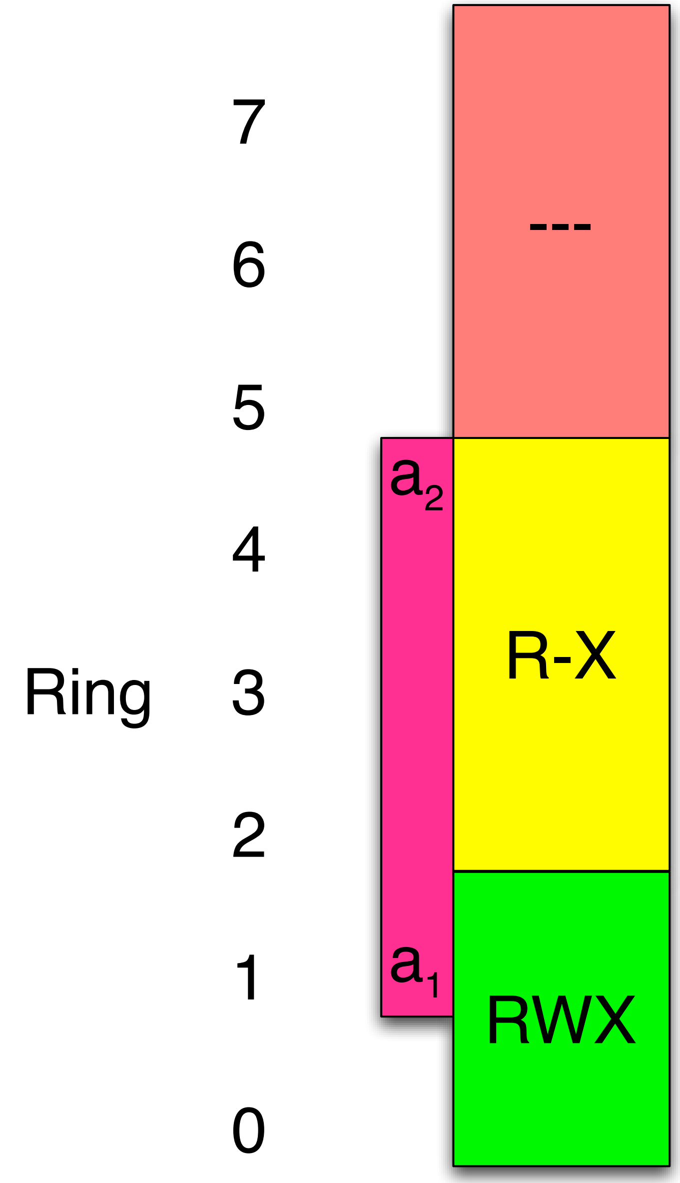
- Program cannot call code of *higher privilege* directly
  - Gate is a special memory address where lower-privilege code can call higher
    - Enables OS to control where applications call it (system calls)



Multics Interpretation

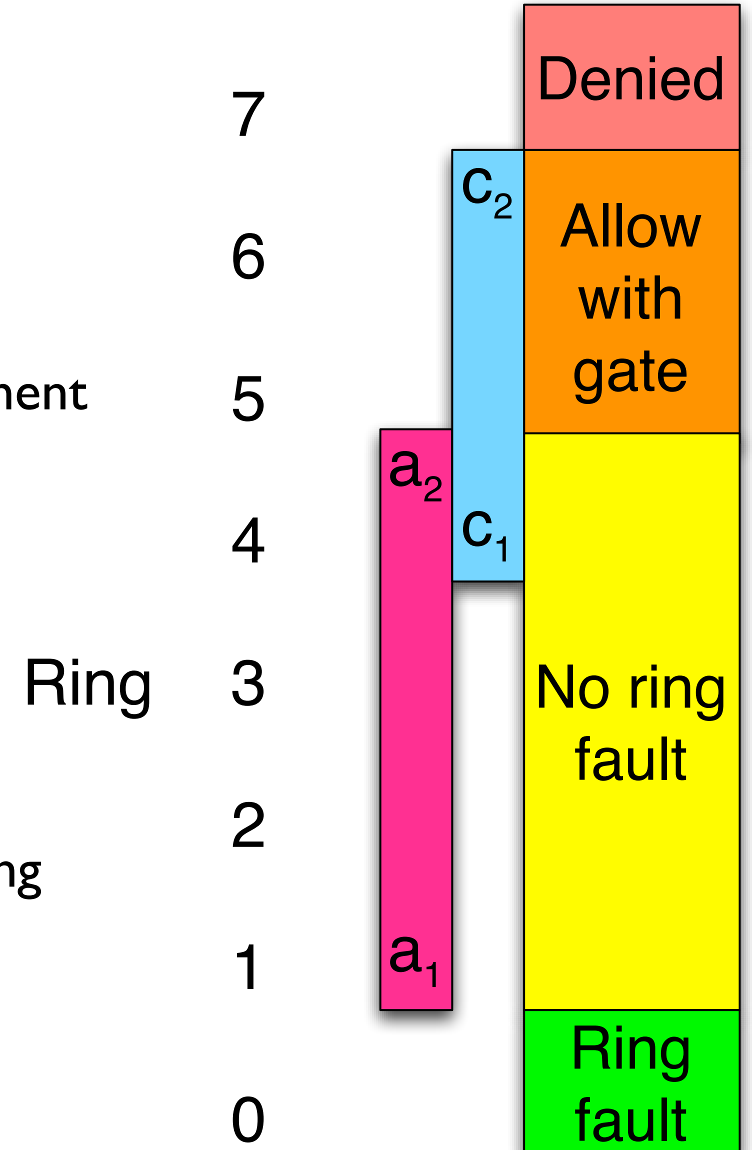
PENNSTATE

- 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 <= a2
    - Describes read and write access to segment
        - 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)

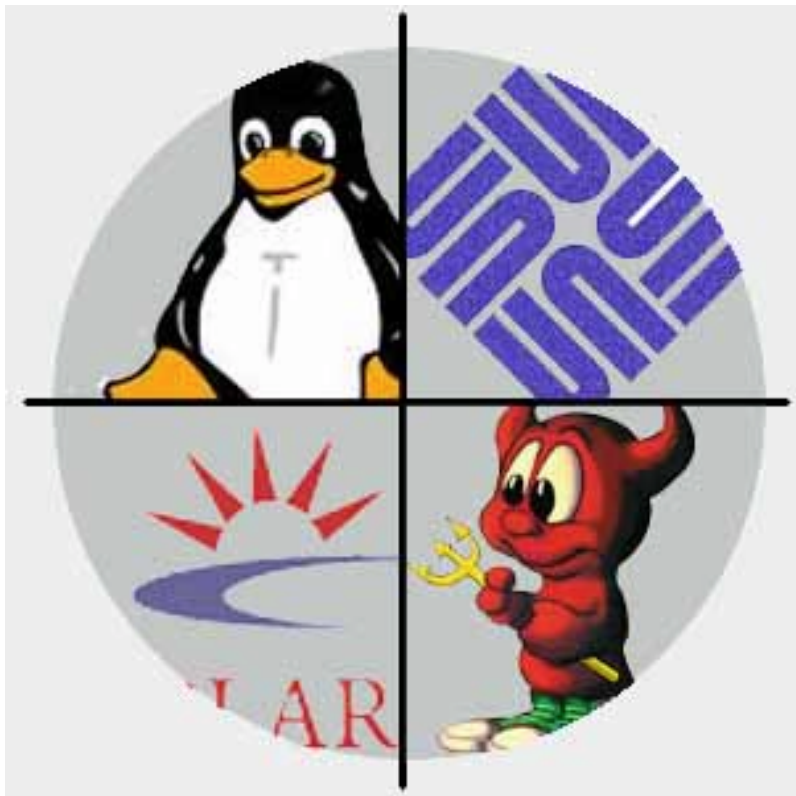
- Also different procedure segments
  - with *call brackets*: (c1, c2), c1 <= c2
  - and access brackets (a1, a2)
  - The following must be true (a2 == c1)
  - Rights to execute code in a new procedure segment
    - r < a1: access permitted with ring-crossing fault
    - a1 <= r <= a2 = c1: access permitted and no fault
    - a2 < r <= 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
- Target code segment defines the new ring



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?
  - How do we determine the new ring?
  - Can new procedure segment access the data segment above?

Now forward to UNIX ...



UNIX Security Limitations

- *Circa 2000 Problems*
  - Discretionary access control
  - Setuid root processes
  - Network-facing daemons vulnerable
- What can we do?

UNIX Security Limitations

- *Circa 2000 Problems*
  - Discretionary access control
  - Setuid root processes
  - Network-facing daemons vulnerable
- What can we do?
  - Reference validation mechanism that satisfies reference monitor concept
  - Protection system with mandatory access control (mandatory protection system)

Linux Security Modules

- Reference validation mechanism for Linux
  - Upstreamed in Linux 2.6
  - Support modular enforcement - you choose
    - SELinux, AppArmor, POSIX Capabilities, SMACK, ...
- 150+ authorization hooks
  - Mediate security-sensitive operations on
    - Files, dirs/links, IPC, network, semaphores, shared memory, ...
  - Variety of operations per data type
    - Control access to read of file data and file metadata separately
- Hooks are restrictive

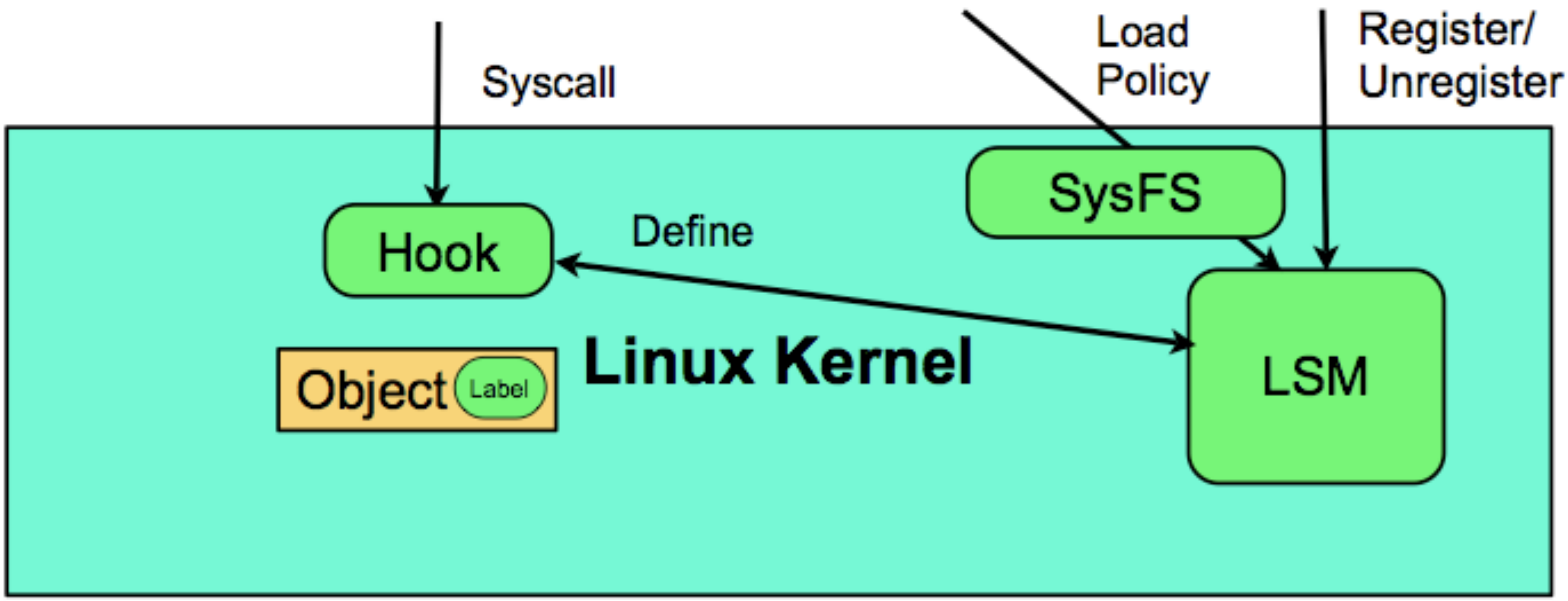
LSM & Reference Monitor

- Does LSM satisfy reference monitor concept?

LSM & Reference Monitor

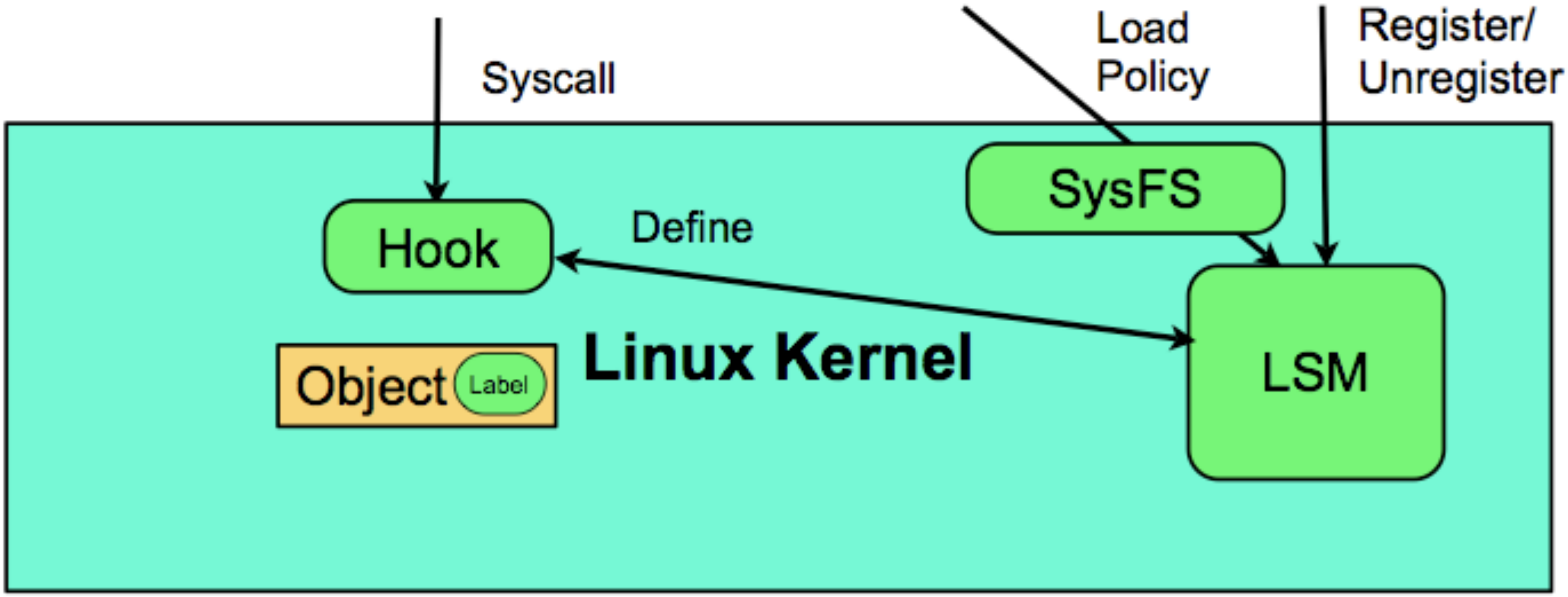
- Does LSM satisfy reference monitor concept?
  - Tamperproof
    - Can MAC policy be tampered?
    - Can kernel be tampered?

Linux Security Modules



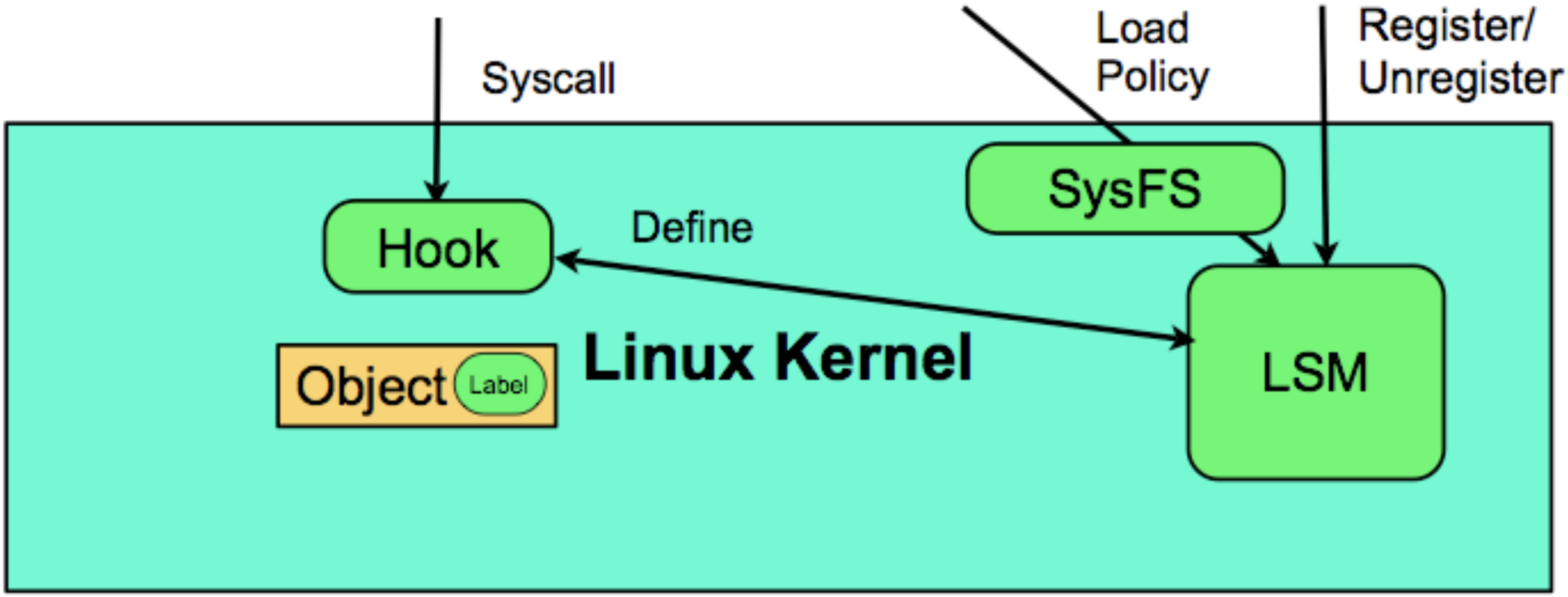
- Register (install) module
- Load policy (open and write to special file)
- Produce authorization queries at hooks

Linux Security Modules



- Attacks on "register"
- Attacks on "install policy"
- Attacks on "system calls"

Linux Security Modules



- To prevent attacks on registration
- And attacks on function pointers of LSM
- LSMs are now statically compiled into the kernel

LSM & Reference Monitor

- Does LSM satisfy reference monitor concept?
  - Tamperproof
    - Can MAC policy be tampered?
    - Can kernel be tampered?
  - Verifiable
    - How large is kernel?
    - Can we perform complete testing?

LSM & Reference Monitor

- Does LSM satisfy reference monitor concept?
  - Tamperproof
    - Can MAC policy be tampered?
    - Can kernel be tampered? By network threats?
  - Verifiable
    - How large is kernel?
    - Can we perform complete testing?
  - Complete Mediation
    - What is a security-sensitive operation?
    - Do we mediate all paths to such operations?

Linux Security Modules

Security check function

```
linux/fs/read_write.c:

ssize_t vfs_read(…) {
    …
    ret = security_file_permission(file, …);
    if (!ret) { …
        ret = file->f_op->read(file, …); …
    }
    …
}
```

Security sensitive operation

LSM & Complete Mediation

- What is a security-sensitive operation?
  - Instructions? Which?
  - Structure member accesses? To what data?
  - Data types whose instances may be controlled
    - Inodes, files, IPCs, tasks, ...
- Approaches
  - Mediation: Check that authorization hook dominates all control-flow paths to structure member access on security-sensitive data type
  - Consistency: Check that every structure member access that is mediated once is always mediated
    - Several bugs found - some years later

LSM & Complete Mediation

- Static analysis of Zhang, Edwards, and Jaeger [USENIX Security 2002]
  - Based on a tool called CQUAL
- Found a TOCTTOU bug
  - Authorize filp in *sys_fcntl*
  - But pass fd again to *fcntl_getlk*
  - Many supplementary analyses were necessary to support CQUAL

```
/* from fs/fcntl.c */
long sys_fcntl(unsigned int fd,
               unsigned int cmd,
               unsigned long arg)
{
  struct file * filp;
  ...
  filp = fget(fd);
  ...

  err = security_ops->file_ops
        ->fcntl(filp, cmd, arg);
  ...
  err = do_fcntl(fd, cmd, arg, filp);

  ...
}

static long
do_fcntl(unsigned int fd,
         unsigned int cmd,
         unsigned long arg,
         struct file * filp) {
  ...
  switch(cmd){
    ...
    case F_SETLK:

      err = fcntl_setlk(fd, ...);

    ...
  }
  ...
}

/* from fs/locks.c */
fcntl_getlk(fd, ...) {
  struct file * filp;
  ...

  filp = fget(fd);

  /* operate on filp */
  ...
}
```

Figure 8: Code path from Linux 2.4.9 containing an exploitable type error.

LSM Enforcement

- Several LSMs have been deployed
  - Most prominent: AppArmor, SELinux, Smack, TOMOYO
- The most comprehensive is SELinux
  - Used by RedHat Fedora and some others

LSM Enforcement

- Several LSMs have been deployed
  - Most prominent: AppArmor, SELinux, Smack, TOMOYO
- The most comprehensive is SELinux
  - Created by the NSA - Result of many years work
  - Used by RedHat Fedora and some others

SELinux Policy Rules

- SELinux Rules express an MPS
  - *Protection state – ALLOW subject-label object-label ops*
  - *Labeling state – Assign new objects labels on creation*
  - *Transition state – Define how a process may change label*
- All are defined explicitly
  - Tens of thousands of rules are necessary for a standard Linux distribution
    - Remember, we are ignoring user processes too (other than confining them relative to the system)
- Enforces a Least Privilege Policy

SELinux Transition State

- For user to run passwd program
  - Only passwd should have permission to modify */etc/shadow*
- Need permission to execute the passwd program
  - *allow user_t passwd_exec_t:file execute* *(user can exec /usr/bin/passwd)*
  - *allow user_t passwd_t:process transition* *(user gets passwd perms)*
- Must transition to passwd_t from user_t
  - *allow passwd_t passwd_exec_t:file entrypoint* *(run w/ passwd perms)*
  - *type_transition user_t passwd_exec_t:process passwd_t*
- Passwd can the perform the operation
  - *allow passwd_t shadow_t:file {read write}* (can edit passwd file)

Take Away

- Goal: Build authorization into operating systems
  - Multics and Linux
- Requirements: Reference monitor
  - Satisfy reference monitor concept
- Multics
  - Hierarchical Rings for Protection
  - Call/Access Bracket Policies (in addition to MLS)
- Linux
  - Did not enforce security (*DAC, Setuid, root daemons*)
  - So, the Linux Security Modules framework was added
  - Approximates reference monitor assuming network threats only -- some challenges in ensuring complete mediation