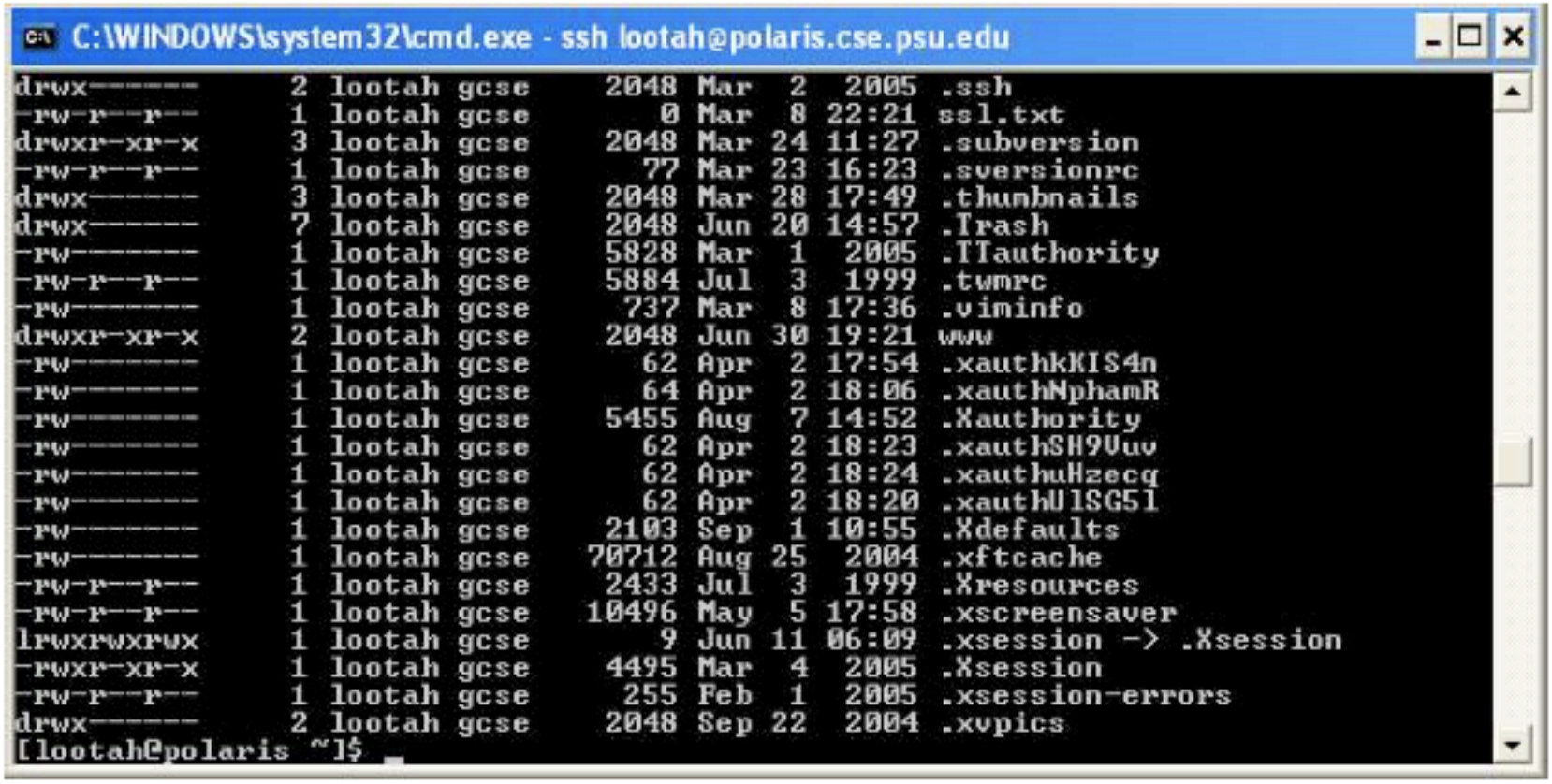
Advanced Systems Security: Linux Security Modules

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

Linux Authorization circa 2000

- Linux implements discretionary access control

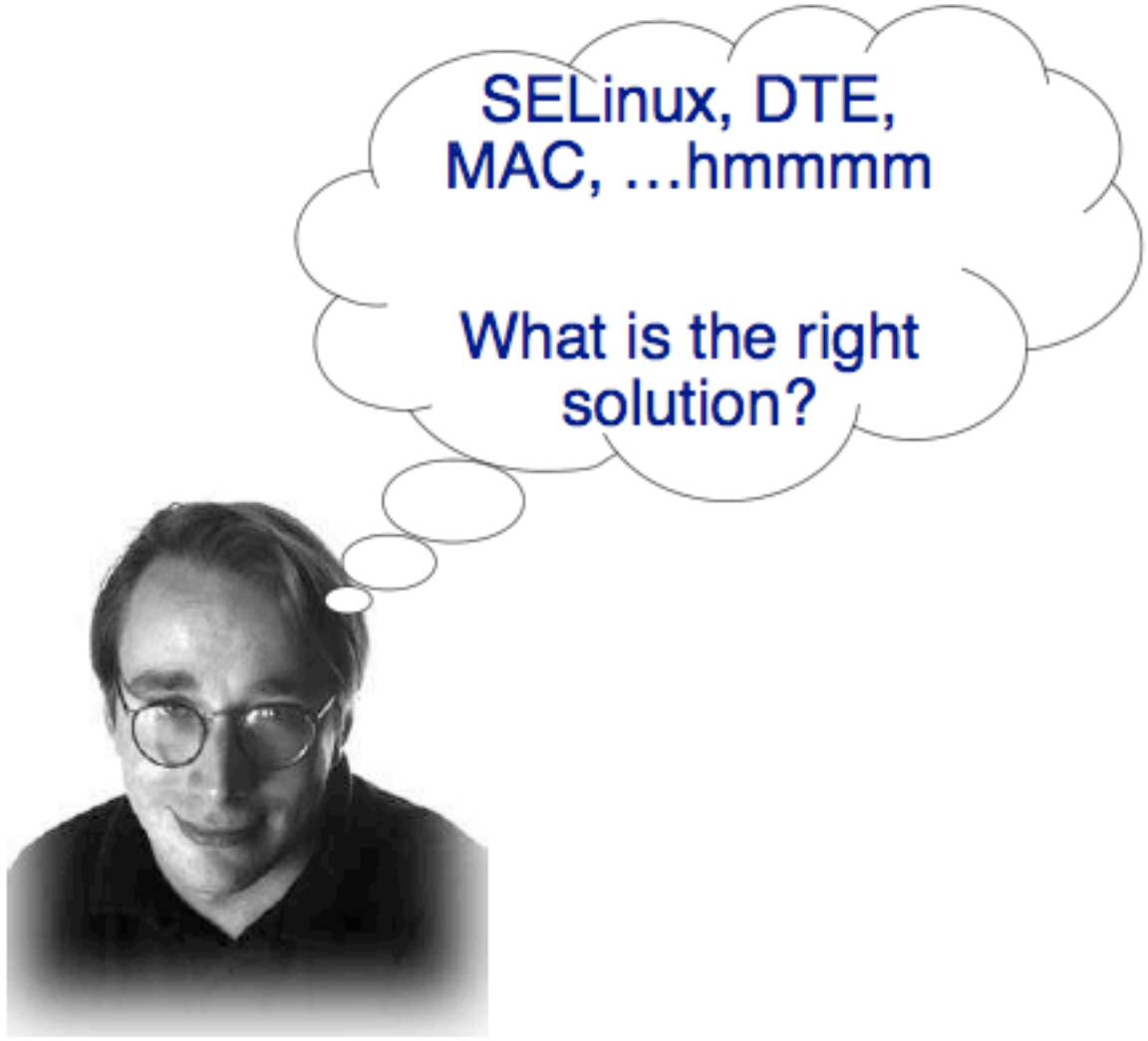


Linux Security circa 2000

PENNSTATE 1855

- Patches to the Linux kernel
  - Enforce different access control policy
    - Restrict root processes
  - Some hardening
- Argus PitBull
  - Limited permissions for root services
- RSBAC
  - MAC enforcement and virus scanning
- grsecurity
  - RBAC MAC system
  - Auditing, buffer overflow prevention, /tmp race protection, etc
- LIDS
  - MAC system for root confinement

Linus' Dilemna



The Answer

- The solution to all computer science problems

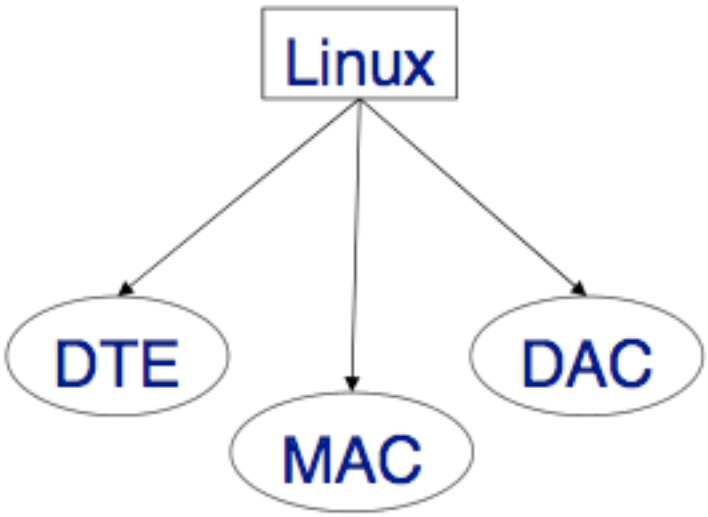
- Add another layer of indirection

Linux Security Modules Was Born

- "to allow Linux to support a variety of security models, so that security developers don't have to have the 'my dog's bigger than your dog' argument, and users can choose the security model that suits their needs.", Crispin Cowan
  - http://mail.wirex.com/pipermail/linux-security-module/2001-April:/0005.html

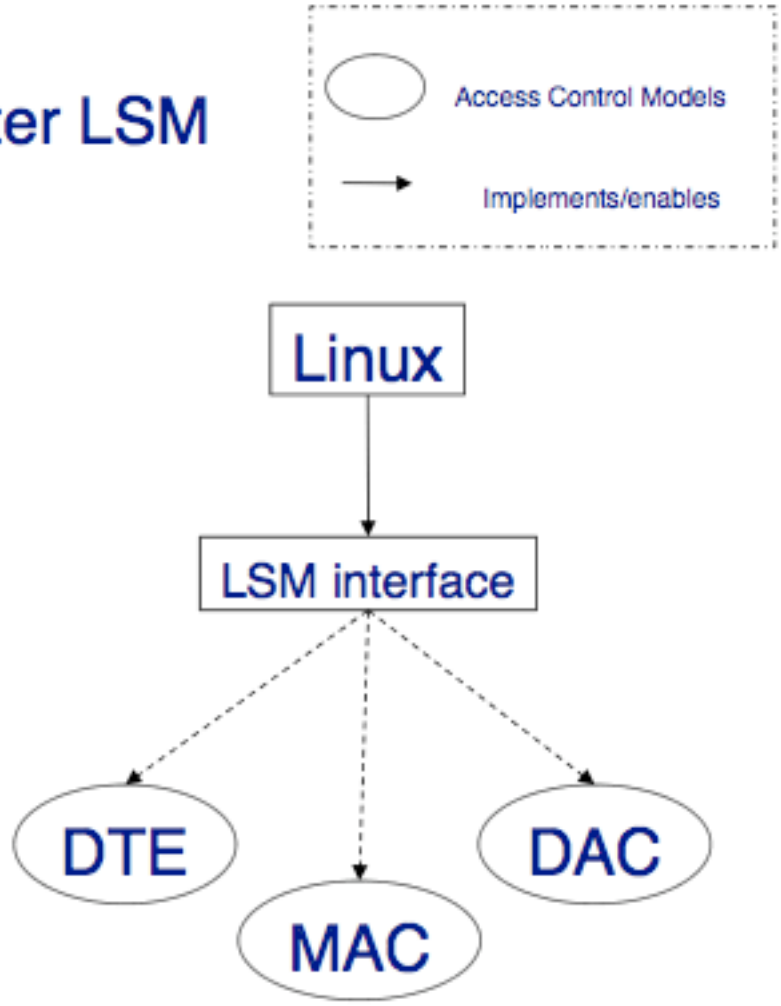
Linux Before and After

Before LSM



Access control models implemented as Kernel patches

After LSM



Access control models implemented as Loadable Kernel Modules

LSM Requirements

PENNSTATE 1855

- LSM needs to reach a balance between kernel developer and security developers requirements. LSM needs to unify the functional needs of as many security projects as possible, while minimizing the impact on the Linux kernel.
  - Truly generic
  - conceptually simple
  - minimally invasive
  - Efficient
  - Support for POSIX capabilities
  - Support the implementation of as many access control models as Loadable Kernel Modules

LSM – A Reference Monitor

- To enforce mandatory access control
  - We need to develop an authorization mechanism that satisfies the reference monitor concept
- How do we do that?
  - And satisfy all the other goals?

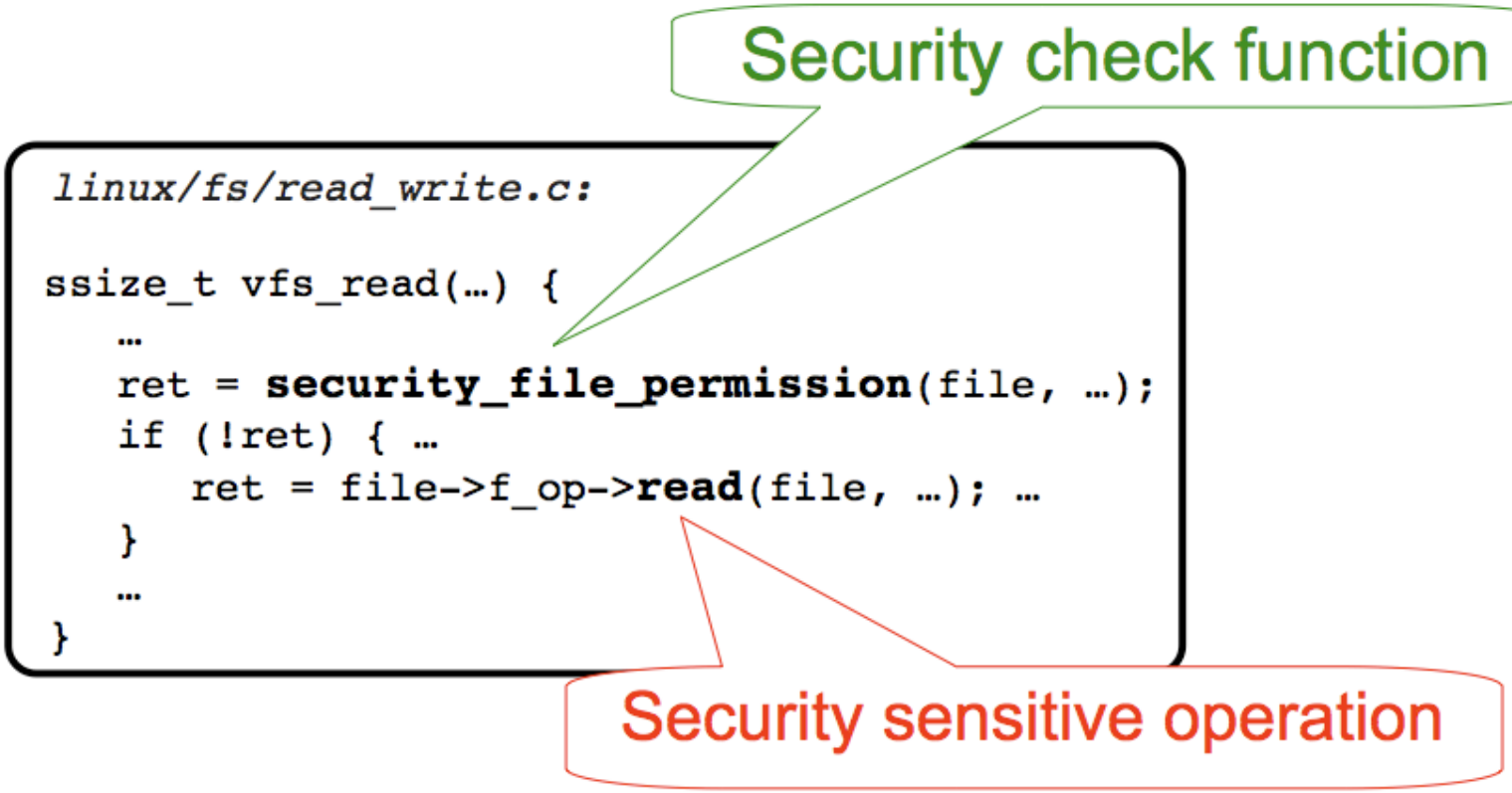
LSM – Complete Mediation

- First requirement is complete mediation
- Add security hooks to mediate various operations in the kernel
  - These hooks invoke functions defined by the chosen module
- These hooks construct "authorization queries" that are passed to the module
  - Subject, Object, Operations

LSM Hooks

- Function calls that can be overridden by security modules to manage security fields and mediate access to Kernel objects.
- Hooks called via function pointers stored in `security->ops` table
- Hooks are primarily “restrictive”

LSM Hooks


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

- First requirement is complete mediation
- Add security hooks to mediate various operations in the kernel
  - These hooks invoke functions defined by the chosen module
- These hooks construct "authorization queries" that are passed to the module
  - Subject, Object, Operations
- Converted to MPS by Module – what does that entail?

LSM – Complete Mediation

- First requirement is complete mediation
- Enables authorization by module
- Linux extends "sensitive data types" with opaque security fields
  - Modules manage these fields – e.g., store security labels
- Which Linux data types are sensitive?

LSM Security Fields

- Enable security modules to associate security information to Kernel objects
- Implemented as void* pointers
- Completely managed by security modules
- What to do about object created before the security module is loaded?

LSM – Complete Mediation

- First requirement is complete mediation
- How do we know LSM implements complete mediation?
- Asked one of the lead developers (Cowan)
  - His reply?

LSM – Complete Mediation

- First requirement is complete mediation
- How do we know LSM implements complete mediation?
- Asked one of the lead developers (Cowan)
  - His reply?
- "We don't"

LSM Analysis

- Static analysis of Zhang, Edwards, and Jaeger [USENIX Security 2002]
  - Based on a tool called CQUAL
- Approach
  - Objects of particular types can be in two states
    - Checked, Unchecked
  - All objects in a "security-sensitive operation" must be checked
    - Structure member access on some types

```
/* Code from fs/read_write.c */
sys_lseek(unsigned int fd, ...)
{
  struct file * file;
  ...
  file = fget(fd);
  retval = security_ops->file_ops
           ->llseek(file);
  if (retval) {
    /* failed check, exit */
    goto bad;
  }
  /* passed check, perform operation */
  retval = llseek(file, ...);
  ...
}
```

LSM Analysis

- Static analysis of Zhang, Edwards, and Jaeger [USENIX Security 2002]
  - Based on a tool called CQUAL
- Found a TOCTTOU vulnerability
  - 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 Analysis

- Runtime analysis of Edwards, Zhang, and Jaeger [ACM CCS 2002]
  - Built a runtime kernel monitor
  - Logs structure member accesses and LSM hook calls
  - Rules describe expected consistency
- Good for finding missing hooks where one is specified
  - Six cases were found

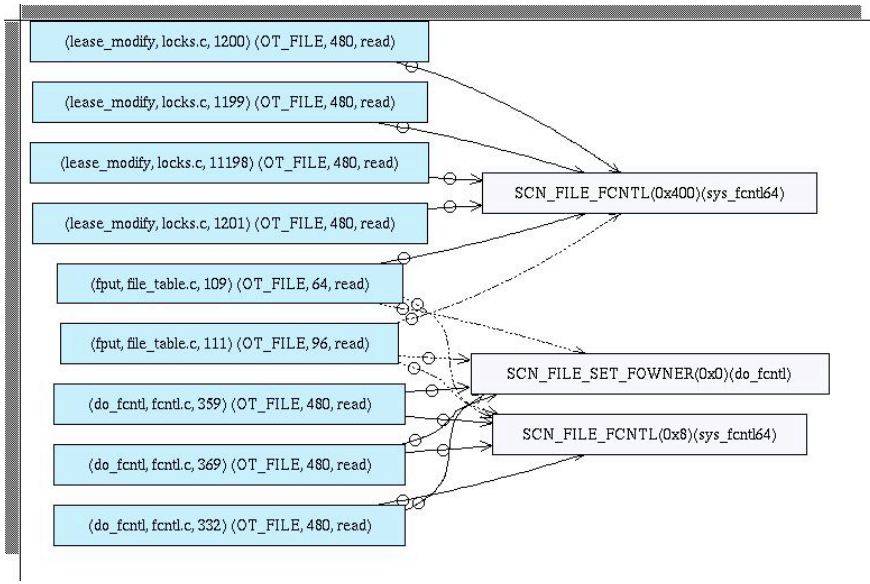
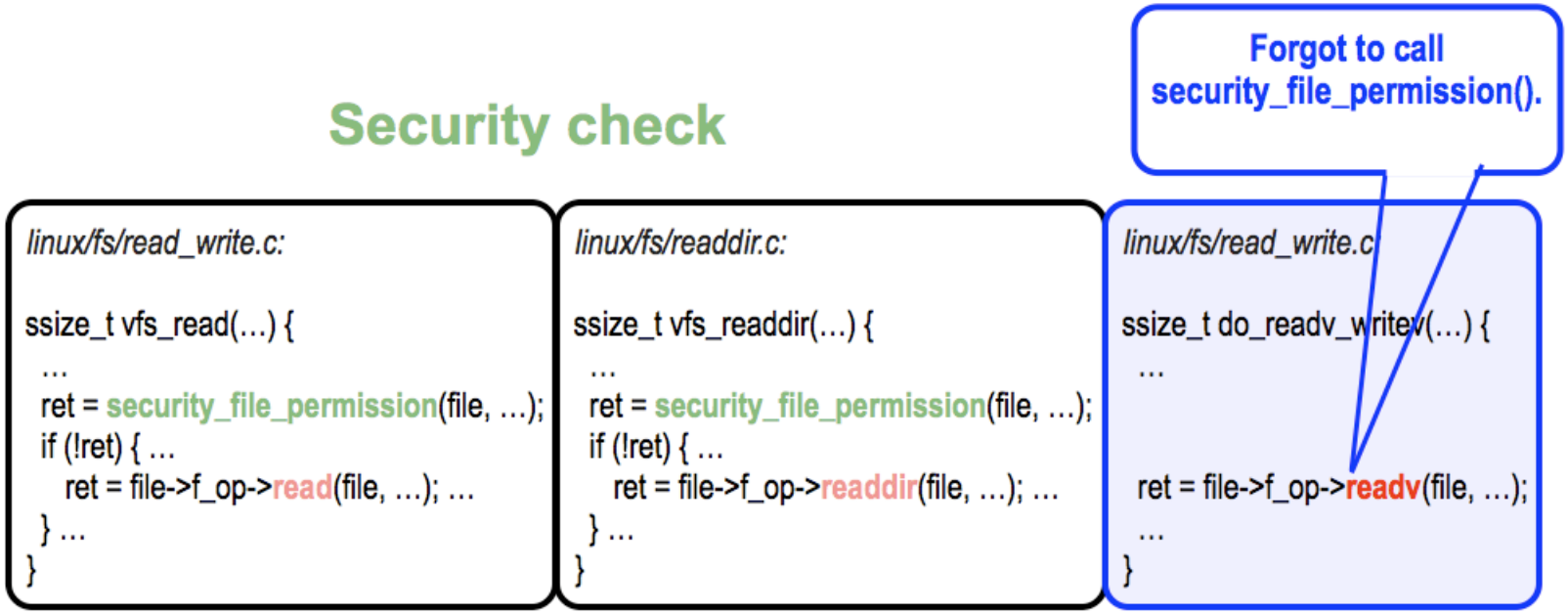


Figure 5: Authorization graph for `fcntl` calls for `F_SETLEASE` (controlled operations in `lease_modify` and `fput`) and `F_SETOWN` (controlled operations in `do_fcntl` and `put`). When command is `F_SETOWN` both `FCNTL` and `SET_OWNER` are authorized, but only `FCNTL` is authorized for `F_SETLEASE`.

LSM Analysis

- Automatically inferring security specifications from code – Tan, Zhang, Ma, Xiong, Zhou [USENIX Security 2008]
  - Automate look at which fns are behind pointers



Same security sensitive operation: file read/write

LSM – Tamperproof

- Second requirement is tamperproof
- Prevent adversaries from modifying the reference monitor code or data
- How is LSM code protected?
- How is LSM data protected?

LSM – Tamperproof

- Second requirement is tamperproof
- Prevent adversaries from modifying the reference monitor code or data
- How is LSM code protected?
- How is LSM data protected?

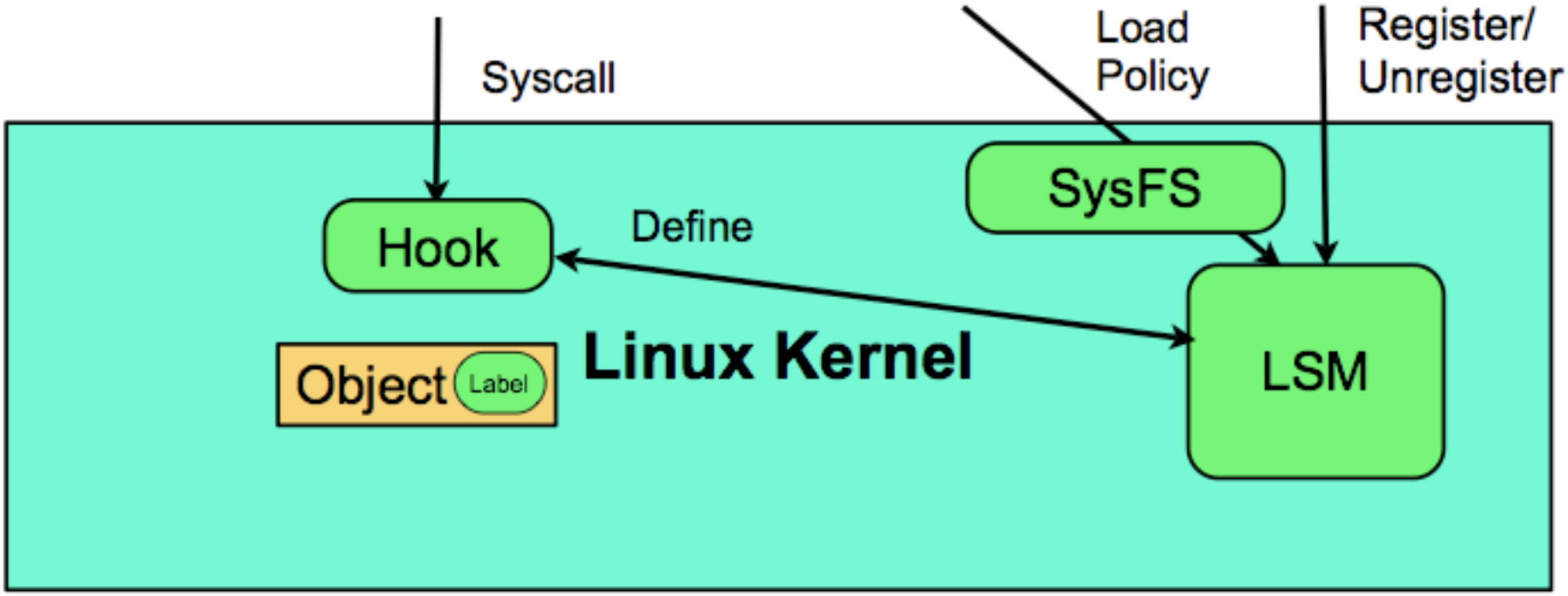
LSM – Tamperproof

- Second requirement is tamperproof
- Add functions to register and unregister Linux Security Modules
  - Implemented as a set of function pointers defined at registration time
- LSM module defines code
- LSM function pointers define targets of hooks
  - These are data – modifiable
- Implications?

LSM – Tamperproof

- Second requirement is tamperproof
- Add functions to register and unregister Linux Security Modules
  - Implemented as a set of function pointers defined at registration time
- Adversaries could modify the code executed by Linux by modifying these function pointer data values
  - Some people opposed this idea and refused to participate
  - Eventually changed to require compiled-in LSM modules

LSM API


Syscall
Load
Policy
Register/
Unregister
SysFS
Hook
Define
Linux Kernel
Object
Label
LSM

LSM Tasks

PENNSTATE 1855

- Linux Kernel modified in 5 ways:
  - Opaque security fields added to certain kernel data structures
  - Security hook function calls inserted at various points with the kernel code
  - A generic security system call added
  - Function to allow modules to register and unregistered as security modules
  - Move capabilities logic into an optional security module

Hook Details

- Difference from discretionary controls
  - More object types
    - 29 different object types
    - Per packet, superblock, shared memory, processes
    - Different types of files
  - Finer-grained operations
    - File: ioctl, create, getattr, setattr, lock, append, unlink,
  - System labeling
    - Not dependent on user
  - Authorization and policy defined by module
    - Not defined by the kernel

LSM Performance

- Microbenchmark: LMBench
  - Compare standard Linux Kernel 2.5.15 with Linux Kernel with LSM patch and a default capabilities module
  - Worst case overhead is 5.1%
- Macrobenchmark: Kernel Compilation
  - Worst case 0.3%
- Macrobenchmark: Webstone
  - With Netfilter hooks 5-7%
  - Uni-Processor 16%
  - SMP 21% overhead

LSM Use

- Available in Linux 2.6
  - Packet-level controls upstreamed in 2.6.16
- Modules
  - POSIX Capabilities module
  - SELinux module
  - Domain and Type Enforcement
  - Openwall, includes grsecurity function
  - LIDS
  - AppArmor
- Not everyone is in favor
  - How does LSM impact system hardening?

Take Away

- Aiming for mandatory controls in Linux
  - But everyone had their own approach
- Linux Security Modules is a general interface for any* authorization module
  - Much finer controls – interface is union of what everyone can do
- What does this effort say about
  - Achieving complete mediation?
  - Whether complete mediation should be policy-dependent?

POSIX Capabilities

PENNSTATE 1855

- POSIX.1e capabilities logic moved into an optional module.
- Capabilities allow partitioning traditional superuser privileges
- Permissive
- Capable interface and task_struct bit vector left as is.