

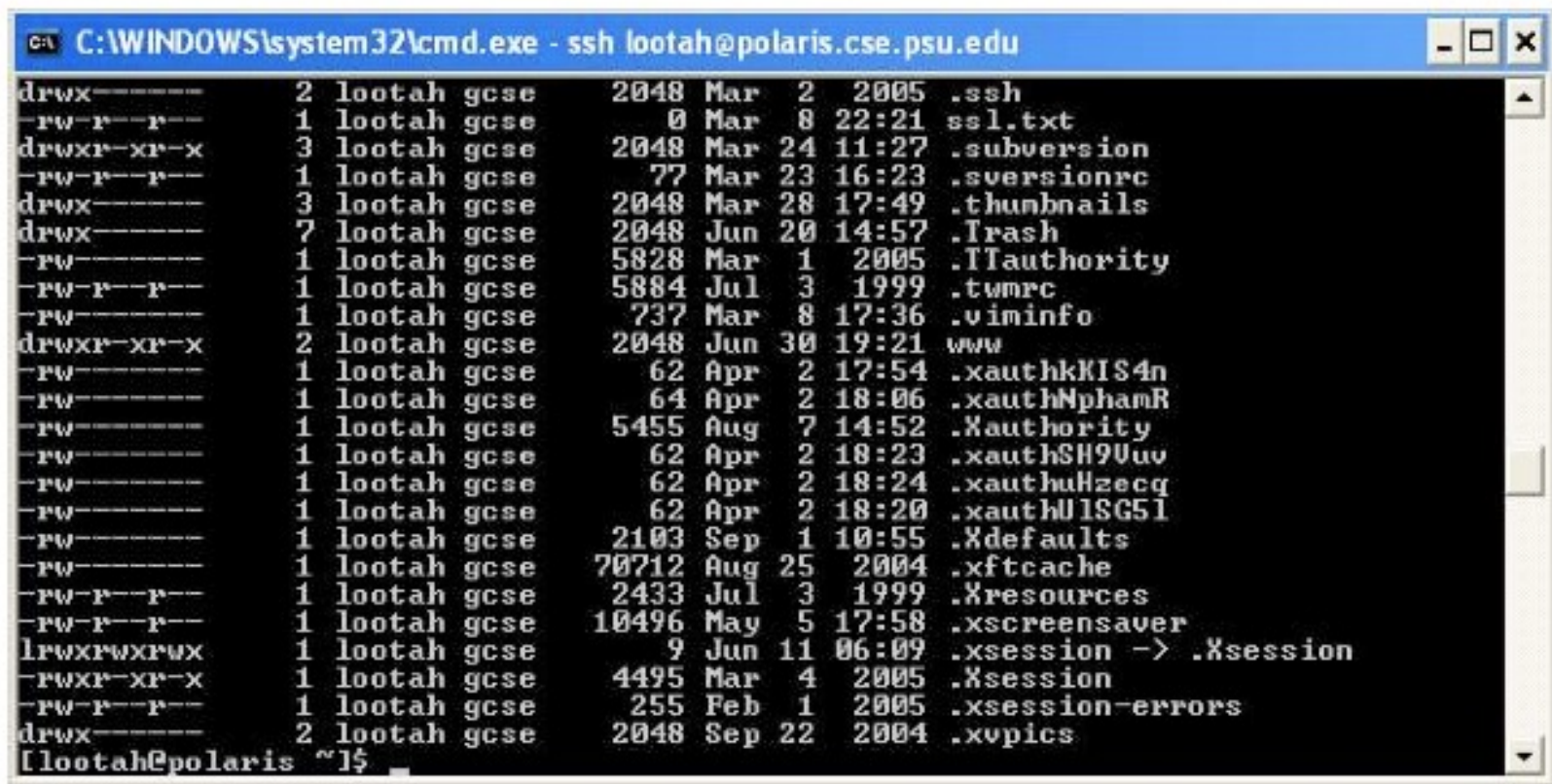
CS260 – Advanced Systems Security

Linux Security Modules

April 16, 2025

Linux Authorization circa 2000

- Linux implements discretionary access control



The screenshot shows a Windows command prompt window titled "C:\WINDOWS\system32\cmd.exe - ssh lootah@polaris.cse.psu.edu". The window displays the output of the 'ls -l' command, showing a list of files and directories with their permissions, owner, group, size, date, time, and name. The files are listed in a table-like format with columns for permissions, owner, group, size, date, time, and name. The permissions are shown in octal notation (e.g., drwxr-xr-x). The owner and group are 'lootah' and 'gcse' respectively. The size is in bytes. The date and time are in MM/DD/YYYY format. The names of the files and directories are listed on the right.

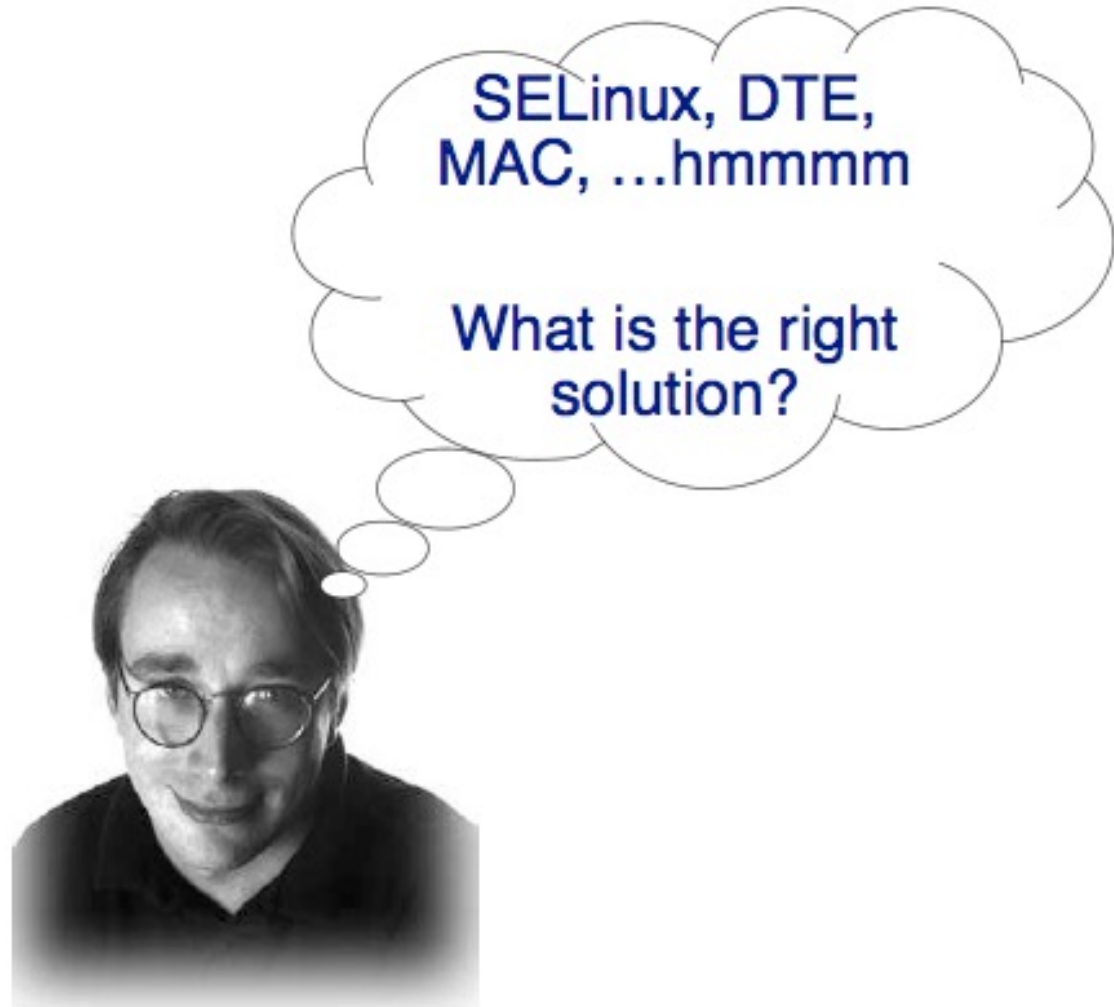
Permissions	Owner	Group	Size	Date	Time	Name
drwxr-xr-x	lootah	gcse	2048	Mar 2	2005	.ssh
-rw-r--r--	lootah	gcse	0	Mar 8	22:21	ssl.txt
drwxr-xr-x	lootah	gcse	2048	Mar 24	11:27	.subversion
-rw-r--r--	lootah	gcse	77	Mar 23	16:23	.sversionrc
drwxr-xr-x	lootah	gcse	2048	Mar 28	17:49	.thumbnails
drwxr-xr-x	lootah	gcse	2048	Jun 20	14:57	.Trash
-rw-r--r--	lootah	gcse	5828	Mar 1	2005	.ITauthority
-rw-r--r--	lootah	gcse	5884	Jul 3	1999	.tvmrc
-rw-r--r--	lootah	gcse	737	Mar 8	17:36	.viminfo
drwxr-xr-x	lootah	gcse	2048	Jun 30	19:21	www
-rw-r--r--	lootah	gcse	62	Apr 2	17:54	.xauthhKIS4n
-rw-r--r--	lootah	gcse	64	Apr 2	18:06	.xauthhNphamR
-rw-r--r--	lootah	gcse	5455	Aug 7	14:52	.Xauthority
-rw-r--r--	lootah	gcse	62	Apr 2	18:23	.xauthhSH9Uuv
-rw-r--r--	lootah	gcse	62	Apr 2	18:24	.xauthhHzecq
-rw-r--r--	lootah	gcse	62	Apr 2	18:20	.xauthhUSG5l
-rw-r--r--	lootah	gcse	2103	Sep 1	10:55	.Xdefaults
-rw-r--r--	lootah	gcse	70712	Aug 25	2004	.xftcache
-rw-r--r--	lootah	gcse	2433	Jul 3	1999	.Xresources
-rw-r--r--	lootah	gcse	10496	May 5	17:58	.xscreensaver
lrwxrwxrwx	lootah	gcse	9	Jun 11	06:09	.xsession -> .Xsession
-rwxr-xr-x	lootah	gcse	4495	Mar 4	2005	.Xsession
-rw-r--r--	lootah	gcse	255	Feb 1	2005	.xsession-errors
drwxr-xr-x	lootah	gcse	2048	Sep 22	2004	.xvpics

[lootah@polaris ~]\$

Linux Security circa 2000

- 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' Dilemma



The Answer

- The solution to all computer science problems

The Answer



- The solution to all computer science problems
 - ▣ Add another layer of indirection

Linux Security Modules Was Born

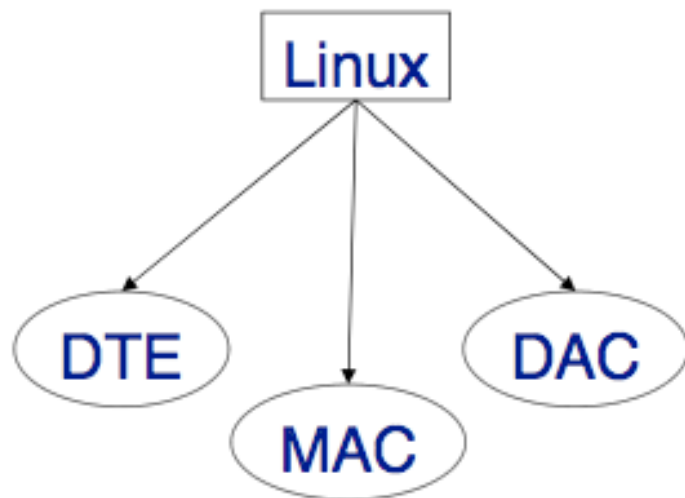


- “to allow Linux to support a variety of security models, so that security developers don't have to have the ‘my dog's bigger than your dog’ argument, and users can choose the security model that suits their needs.”, Crispin Cowan

– <http://mail.wirex.com/pipermail/linux-security-module/2001-April/0005.html>

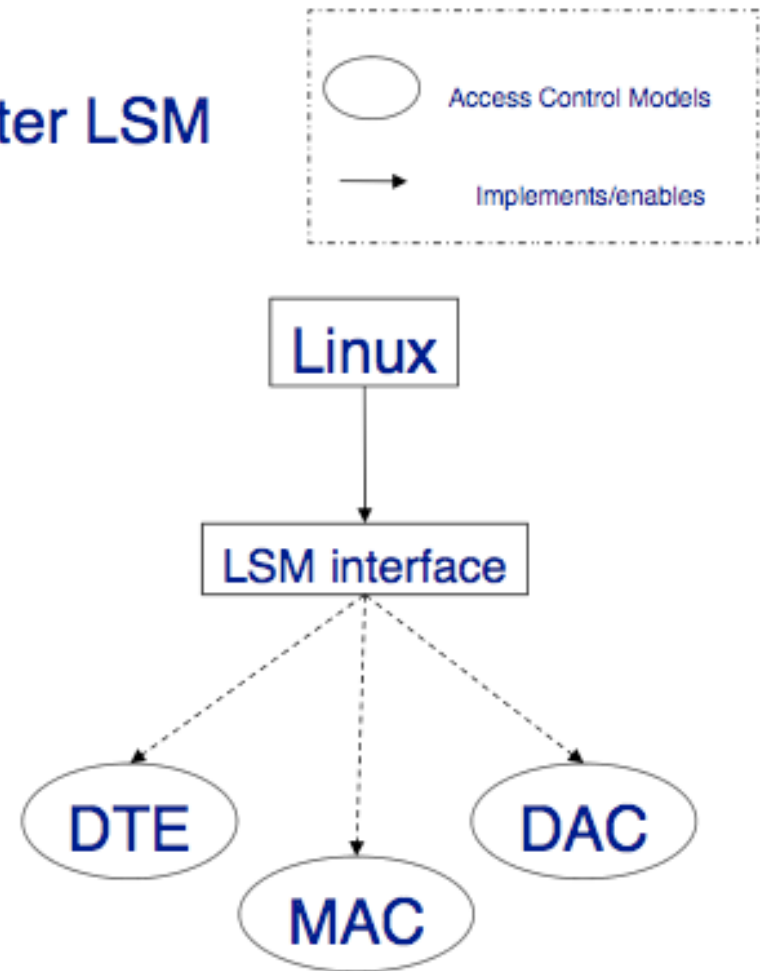
Linux Before and After

Before LSM



Access control models implemented as
Kernel patches

After LSM



Access control models implemented as
Loadable Kernel Modules

LSM Requirements



- 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

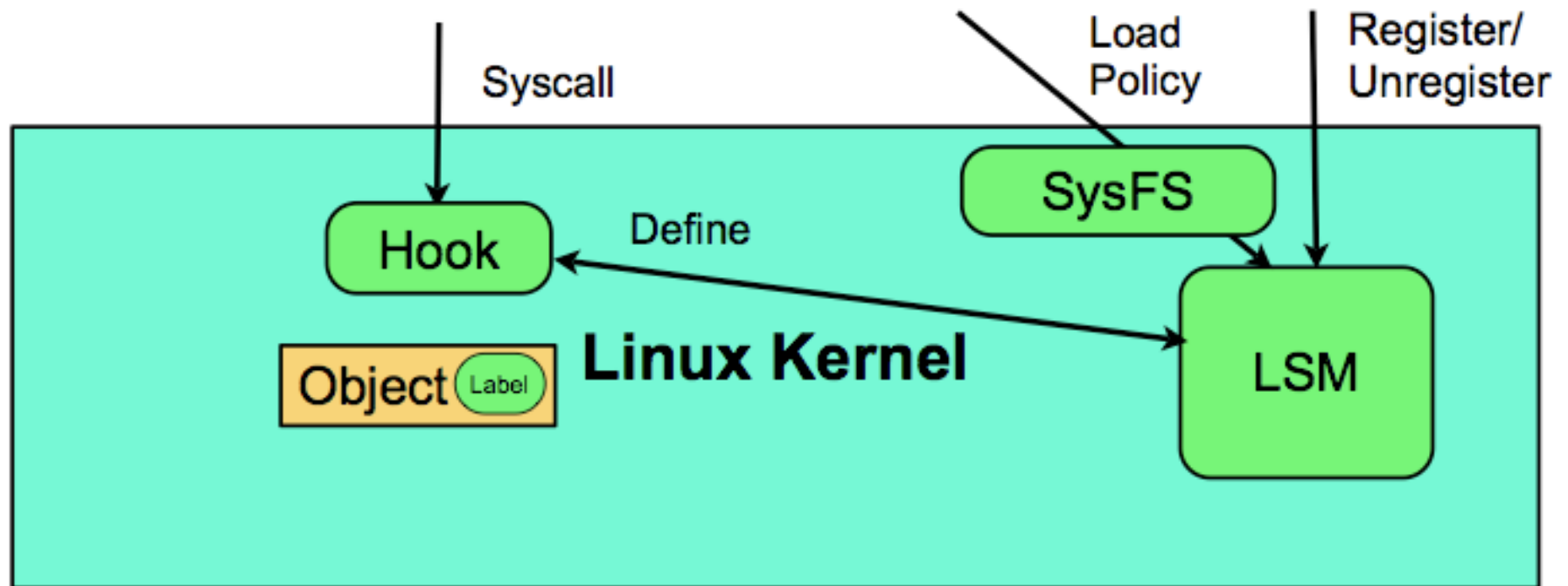
- Function calls that can be overridden by security modules to manage security fields and mediate access to Kernel objects.
- Hooks called via function pointers stored in `security->ops` table
- Hooks are primarily “restrictive”

LSM Security Policy



- 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 API



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**
- 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 – 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 (1)

- **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/readwrite.c */
sys_llseek(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 (1)

- 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
- Good for finding missing hooks

```
/* 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 (2)

- 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 when 2+ hooks are expected
 - ▣ Six cases were found

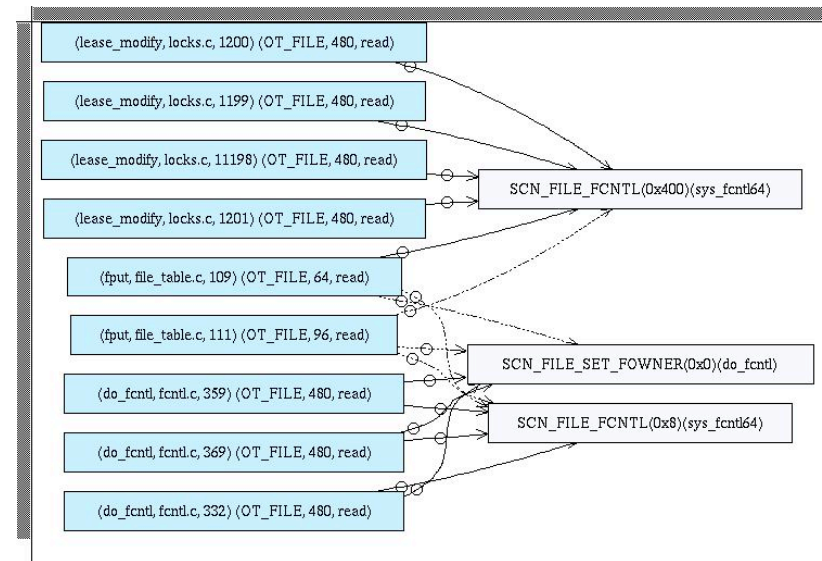


Figure 5: Authorization graph for `fcntl` calls for `F.SETLEASE` (controlled operations in `lease_modify` and `fput`) and `F.SETOWN` (controlled operations in `do_fcntl` and `put`). When command is `F.SETOWN` both `FCNTL` and `SET_OWNER` are authorized, but only `FCNTL` is authorized for `F.SETLEASE`.

LSM Analysis (3)

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

Security check

linux/fs/read_write.c:

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

linux/fs/readdir.c:

```
ssize_t vfs_readdir(...) {  
    ...  
    ret = security_file_permission(file, ...);  
    if (!ret) { ...  
        ret = file->f_op->readdir(file, ...); ...  
    } ...  
}
```

linux/fs/read_write.c:

```
ssize_t do_readv_writev(...) {  
    ...  
    ret = file->f_op->readv(file, ...);  
    ...  
}
```

Forgot to call
security_file_permission().

Same security sensitive operation: file read/write

LSM – Tamperproof



- Second requirement is **tamperproofing**
- 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

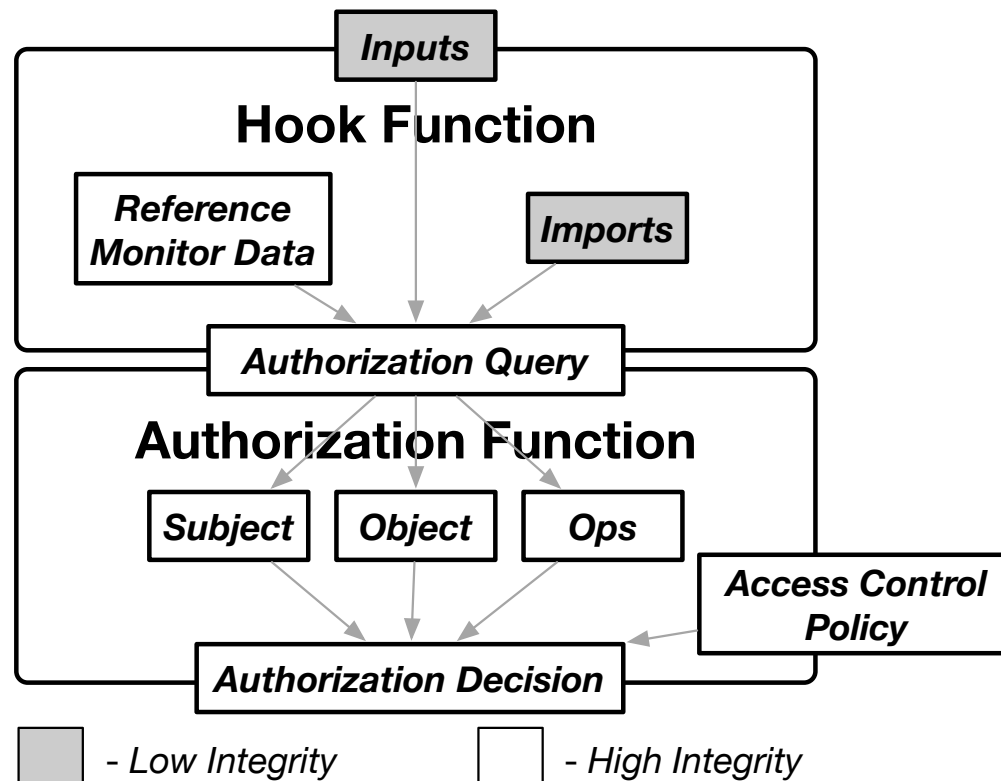
- Second requirement is **tamperproof**
- Add functions to register and unregister Linux Security Modules
 - ▣ Implemented as a set of function pointers defined at registration time
- Adversaries could modify the code executed by Linux by **modifying these function pointer data values**
 - ▣ Some people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

LSM – Tamperproof



- Second requirement is **tamperproof**
- Anything else that an adversary could modify?

LSM - Tamperproof



LSM – Tamper Analysis

Talisman, NDSS 2024

```
1 static int inode_has_perm(const struct cred *cred,  
    struct inode *inode, u32 perms, struct  
    common_audit_data *adp) {  
2     ...  
3     validate_creds(cred);  
4     if (unlikely(IS_PRIVATE(inode)))  
5         return 0;  
6  
7     sid = cred_sid(cred);  
8     isec = inode->i_security;  
9     return avc_has_perm(sid, isec->sid, isec->sclass,  
        perms, adp);  
10 }
```

Fig. 1. Function `inode_has_perm` from SELinux

Security metadata is stored with objects – see Line 8

```
1 static int selinux_file_open(struct file *file,  
2     const struct cred *cred) {  
3     struct file_security_struct *fsec;  
4     struct inode_security_struct *isec;  
5  
6     fsec = file->f_security;  
7     isec = file_inode(file)->i_security;  
8  
9     /* Endorse: verify task */  
10    __u64 val;  
11    val = EXX_VALUE_SELINUX_TASK(tsec);  
12    exx_verify(&exx_se_task, EXX_KEY_TASK(get_current()),  
        &val, sizeof(val));  
13  
14    /* Endorse: verify inode */  
15    val = EXX_VALUE_SELINUX_INODE(isec);  
16    exx_verify(&exx_se_inode, EXX_KEY_INODE(isec->inode),  
        &val, sizeof(val));  
17  
18    fsec->isid = isec->sid;  
19    fsec->pseqno = avc_policy_seqno();  
20  
21    /* Endorse: add file */  
22    void *dup = exx_dup((void *) fsec, sizeof(*fsec));  
23    if (dup)  
24        exx_add(&exx_se_file, EXX_KEY_FILE(file), dup,  
            sizeof(*fsec));  
25  
26    return file_path_has_perm(cred, file, open_file_to_av(  
        file));  
27 }  
~o
```

Endorse use of security metadata
- E.g., Values have not changed since the object was created

LSM Tasks



- 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 unregister 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?

Questions

36

