CIS 610: Advanced Topics in Systems Security
Linux Security Modules

Prof. Kevin Butler

Fall 2011
Notes

• Class November 7
  ‣ Likely back on
  ‣ Midterm: in-class vs takehome?

• Project
  ‣ Related work for project
  ‣ Project Experiment proposal
  ‣ Project Deadline

• Assignment 1
  ‣ due in one week
Linux Authorization circa 2000

- Linux implements discretionary access control
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’s Dilemma

SELinux, DTE, MAC, ...hmmmm

What is the right solution?
The Answer

• The solution to all computer science problems

• Add another layer of indirection
Linux Before and After

Before LSM

Linux
- DTE
- DAC
- MAC

Access control models implemented as Kernel patches

After LSM

Linux
- LSM interface
  - DTE
  - DAC
  - MAC

Access control models implemented as Loadable Kernel Modules
Linux Security Modules Were 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

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 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 unregistered as security modules
  - Move capabilities logic into an optional security module
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 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

```
linux/fs/read_write.c:

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

- 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.
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 Hook Architecture

Process

open System Call

Lookup inode

Process file path down to inode (resolving directories/links)

DAC checks

"Is user_process allowed to perform operation on inode?"

LSM hook

Access inode

LSM Policy Engine

User Space

Kernel Space
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?
LSM and Complete Mediation

- LSM is mainly responsible for complete mediation
  - What was the basis for choosing security-sensitive operations?
  - Pretty ad hoc…

- How did that work out?
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 “controlled operation” must be checked
    - Structure member access on objects

```c
/* 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 bug
  - Authorize filp in `sys_fcntl`
  - But pass fd again to `fcntl_getlk`
- Many supplementary analyses were necessary to support CQUAL

```c
/* 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
  - Rules describe expected consistency

- Good for finding missing hooks where one is specified
  - Six cases were found
LSM Analysis

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

![Security check](image.png)

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_readdv_writer(...) {
    ...
    ret = file->f_op->readdv(file, ...);
    ...
}
```

Forgot to call `security_file_permission()`.

Same security sensitive operation: `file_read/write`
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?