

CS 378 (Spring 2003)

Linux Kernel Programming

Yongguang Zhang

(ygz@cs.utexas.edu)

This Lecture

- Linux Security (kernel)
 - Capabilities
 - LSM
- Questions?

Capabilities

- Fine grain control of who can do what
 - Traditional: all-or-nothing: super user (root) can do everything, normal user (non-root) can do nothing
 - Capabilities: define a set of distinct privileges in the system (if a task has a capability, it is permitted to do certain action)
- Definition
 - POSIX 1.e defines a list of capabilities
 - Linux 2.4 implements 8 from POSIX, and adds 21 Linux-specifics (total 29)

Capability Definitions

- All definitions in `include/linux/capability.h`
 - `CAP_CHOWN`: allow changing file ownership
 - `CAP_SETUID`: allow making `setuid()` call
 - `CAP_SETPCAP`: allow modifying other task's capability (under the same set of capabilities it owns)
 - `CAP_NET_BIND_SERVICE`: allow binding to TCP/UDP port below 1024
 - `CAP_NET_BROADCAST`: allow broadcasting
 - `CAP_NET_RAW`: allow raw sockets
 - `CAP_SYS_NICE`: allow changing nice level
 - ...

Data Type for Capability Set

- A set of capabilities: `kernel_cap_t`
 - Defined in `include/linux/capability.h`
 - 32-bit integer
 - Bitmap: 1 bit per capability: 1 means having the corresponding capability, 0 means no
 - Maximum 32 capabilities support in Linux 2.4
- Operations:
 - `cap_raise(c, flag)`: Include the capability in `c`
 - `cap_lower(c, flag)`: Remove the capability from `c`
 - `cap_raised(c, flag)`: `c` having the capability?

Capability Sets in Task

- Each task has 3 sets of capabilities
 - In struct `task_struct`, fields related to capabilities:
`kernel_cap_t cap_effective, cap_inheritable, cap_permitted;`
- POSIX capability model
 - Permitted set: capabilities the task can use
 - Effective set: capabilities that the task currently choose to use (so as to bracket operations)
 - Inheritable set: capabilities that are allowed through program execution (via `execve()` system call)

Use Capabilities

- Kernel can check the capability before doing privileged action:

```
...  
if (!capable(CAP_XXX))  
    return -EPERM;  
...
```

- **capable(cap):** does this task has the capability?

```
int capable(int cap)  
{  
    if (cap_raised(current->cap_effective, cap))  
        return 1;  
    return 0;  
}
```

Capability Example

- Controlling system call nice()
 - In kernel/sched.c:

```
asmlink long sys_nice(int increment)
{

    if (increment < 0) {
        if (!capable(CAP_SYS_NICE))
            return -EPERM;
```

File Capabilities

- Executable files can have capabilities too
 - Also have 3 sets: permitted, effective, inheritable
 - Theoretically stored as file attributes in file systems
 - Change the task's capabilities after `execve()`
- Capability rules
 - Inheritable set does not change after `execve()`
 - New permitted set = file permitted set OR (file inheritable set AND task permitted set)
 - New effective set = file effective set AND new permitted set
 - Implementation: `compute_creds()` (`fs/exec.c`)

File Capability Implementation

- Executable file data structure `struct linux_binprm`
 - Defined in `include/linux/binfmts.h`
 - Fields related to capabilities:
`kernel_cap_t cap_inheritable, cap_permitted, cap_effective;`
- When an executable file is loaded:
 - Fill in `linux_binprm` from file system and call `compute_creds()`
 - Example: load ELF file: function `load_elf_binary()`
- Reality check
 - Linux 2.4 VFS does not support capability attributes

New Direction in Security Kernel

- Linux 2.5: LSM (Linux Security Modules)
 - Fine-grained nondiscretionary access controls
 - General kernel framework for implementing security modules
- Goal
 - To replace the Linux 2.4 capability model
 - Pluggable kernel modules to provide different security models: superuser logic, POSIX capability logic, SELinux model, etc.
- LSM project: <http://lsm.immunix.org>

LSM Basics

- Add a “security” field to major data structures:
 - task_struct, super_block, inode, file, sk_buff, net_device, ...
 - Type: void *security;
- Add hooks in kernel critical points
 - To manage the security field
 - To perform access control
- Security module register and unregister functions
- New security() system call

Global Security Ops Table

- A table of all security operations (127)
 - Stored in a global variable `security_ops`
 - Defined in `include/linux/security.h`
 - Data type:

```
struct security_operations {  
    ...  
    int (*sb_mount)(...);  
    ...  
    int (*inode_unlink)(...);  
    ...  
    int (*task_setnice)(...);  
    ...  
};
```

security_ops Example

- Controlling system call nice()
 - In kernel/sched.c:

```
asmlink long sys_nice(int increment)
{
    ...
    retval = security_task_setnice(current, nice);
    if (retval)
        return retval;
    ...
}
```
 - security_task_setnice(p, nice) defined in include/linux/security.h to be

```
return security_ops->task_setnice(p,nice);
```

LSM Modules

- How to write a new LSM module:
 - Define a `security_operations` data object
 - Fill in all 127 functions
 - Call `register_security()` in module init and `unregister_security()` in module exit
- Existing LSM modules (in Linux 2.5)
 - In Linux 2.5: Dummy, Capability
 - As a patch: Openwall, SELinux
 - Source code: under `security/`