Operating Systems

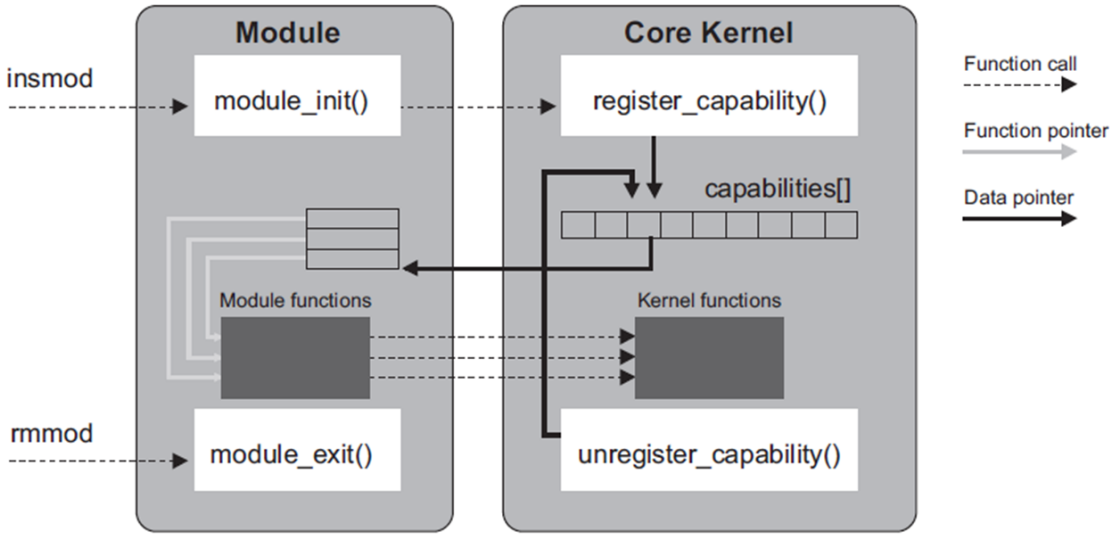
Recitation 11

Plan

- Controlling driver (ioctl)
- Intercepting system call
- Debug-FS
- Keyboard sniffer

Last week we saw a simple character device driver.

Let's add "encryption upon demand" capability.



So far –
Driver registered operations, kernel invoked those operations when needed.

Now –
We want to alter driver behavior, i.e. to "talk" to driver not to the device it controls.

file_operations `ioctl()`

```
int (*ioctl)(struct file *filp,
             unsigned int cmd,
             unsigned long arg);
```

- For special commands not covered by existing functions (read, write,…)
  - Considered bad programming to use it otherwise
- ~~inode – device inode file~~ (Deprecated)
- filp – open device file
- cmd – command number
- arg – optional general purpose
- Returns – depends on implementation, <0 on error

The “TO DO” list:

1. Define the operation - flip “with encryption” flag.
   We need to create a new header file
2. Write code that gets that command and reacts on it.
   Implement and register certain *ioctl* function in driver.
3. Write user space code that raises/drops the flag.
   Use the header file from #1

CHARDEV-2

Device driver changes:

- create header file, predefine MAJOR_NUM
- add device_ioctl function
- register it in file_operations structure

CHARDEV-2

In user space:

- create user_chardev.c

The only thing is does:

invokes ioctl function for our device file,

sets encryption flag to 1.

The flow

[9228.487115] device_open(d5d85780)
[9227.487150] device_write(d5d85780,8)
[9227.487161] device_release(f308cb58,d5d85780)
[9230.357332] device_open(f4c2fd80)
[9230.357368] device_read(f4c2fd80,131072) - operation not supported yet **(last written - Hello OS)**
[9230.358490] device_release(f308cb58,f4c2fd80)
[9320.436419] device_open(d7639840)
[9320.436438**] chardev, ioctl: setting encryption flag to 1**
[9320.436444] device_release(f308cb58,d7639840)
[9335.459575] device_open(d7639840)
[9335.459616] device_write(d7639840,8)
[9335.459627] device_release(f308cb58,d7639840)
[9338.669655] device_open(f53dfc00)
[9338.669693] device_read(f53dfc00,131072) - operation not supported yet **(last written - Ifmmp!PT)**
[9338.670776] device_release(f308cb58,f53dfc00)

Example – CHARDE2

Intercepting system call

- Kernel exports system calls table

  cat /proc/kallsyms | grep sys_call_table
- It is an array of pointers to functions
- Let’s substitute one of them

Problems:
It is hidden.
It is write protected.

Intercepting system call

Solution:

Take an address of a known function (sys_close)
Iterate through kernel memory,
Check
if current position can be considered
as the sys_call_table,
then the certain entry contains
the pointer to that known function

Intercepting system call

Once sys_call_table found,

Access control register cr0 (in CPU)
Make the page writable
Preserve the pointer to the original syscall function
Assign the pointer to our new function
Restore read-only mode

Intercepting system call

Upon system call:

Do additional things (printk)

Invoke original system call

Intercepting system call

Cleaning up

Access control register cr0

Make page writable

Restore the pointer to the original syscall function

Restore read-only mode

Example - Interceptor

Debug-FS

debugfs is a simple memory-based filesystem, designed specifically to debug Linux kernel code

Setting up:

mount -t debugfs nodev /sys/kernel/debug

(Probably done already)

Debug-FS

The typical sequence

On init
debugfs_create_dir
debugfs_create_file

On some event:
pr_debug

On cleanup
debugfs_remove_recursive

Keyboard sniffer

Problem:

- Standard file_operations are not enough
- Too much information to put into a standard log with printk

Solution:

Call register_keyboard_notifier to register certain callbacks.

Use debugfs to store all the keys pressed

Keyboard sniffer

Linux uses **notification chains**.

These notifier chains work in a **Publish-Subscribe model**. This model is more effective when compared to polling or the request-reply model.

Descriptor structure -

struct notifier_block

Example - Keysniffer

Keyboard sniffer

register_keyboard_notifier

start sniffing

unregister_keyboard_notifier

remove the sniffer

Example - keysniffer