# Linux Kernel Development - Introduction to kernel modules

*2012-7-13, Open Lab Summer Student Lectures*

**Panos Sakkos**
**CERN Security Team -** http://cern.ch/security
Technical Student, National and Kapodistrian University of Athens

CERN IT Department
CH-1211 Genève 23
Switzerland
**www.cern.ch/it**

# Overview



- Work of CERN Security Team (CST) within the kernel
- Architecture of an OS
- Anatomy of a kernel module
- kprobes
- Debugging
- Good and Bad practices

# Work of CST within the kernel

- netlog

# netlog

- **Logs information for every connection on the hosted machine**
  - Who, connected where and when
    - Date and time
    - hostname
    - process
    - user
    - local/remote ip addresses and ports
    - Protocol of connection
    - Action (connect, accept, close, bind)
  - Traceability++
  - Open source http://cern-cert.github.com/netlog

CERN IT Department
CH-1211 Genève 23
Switzerland
**www.cern.ch/it**

```
[root@kmod-testing netlog]# !
[root@kmod-testing netlog]# tail /var/log/messages -n 9
Jul 11 14:18:22 kmod-testing kernel: netlog: Light monitoring tool for inet connections by CERN Security Team
Jul 11 14:18:22 kmod-testing kernel: netlog:    [+] Planted connect pre handler
Jul 11 14:18:22 kmod-testing kernel: netlog:    [+] Planted connect post handler
Jul 11 14:18:22 kmod-testing kernel: netlog:    [+] Planted accept post handler
Jul 11 14:18:22 kmod-testing kernel: netlog:    [+] Planted close pre handler
Jul 11 14:18:22 kmod-testing kernel: netlog:    [-] Absolute path mode is disabled. The logs will contain the pr
ocess name
Jul 11 14:18:22 kmod-testing kernel: netlog:    [+] Deployed
Jul 11 14:18:29 kmod-testing kernel: netlog: ssh[4707] TCP 128.141.48.75:38530 -> 137.138.141.158:22 (uid=0)
Jul 11 14:18:30 kmod-testing kernel: netlog: firefox[3914] TCP 128.141.48.75:55168 <-> 137.138.144.172:443 (uid=
0)
```

# Work of CST within the kernel

- netlog
- tty-kraven

# tty-kraven

- **Detects tty hijacks**
  - First (known) tool that detects this kind of attack
  - Logs
    - Injected command
    - the victim TTY driver name
    - the executable of the attacker
    - PID of the attacking process

```
[root@kmod-testing test_cases]# tail /var/log/messages -n 3
Jul 11 14:21:29 kmod-testing kernel: tty-kraven: tty hijack detection tool from CERN Security Team
Jul 11 14:21:29 kmod-testing kernel: tty-kraven:        [+] Planted
Jul 11 14:22:04 kmod-testing kernel: tty-kraven: 'passwd' to [/dev/pts/1] from (/root/kraven/test_cases/a.out) p
id: 5346
```

# Work of CST within the kernel

- netlog
- tty-kraven
- redeemer

# redeemer

- **rootkit detector**
  - Designed to detect even rootkits that utilize the debug registers!
    - First know tool to succeed this depth of detection
- **Hashes and checks parts of the kernel**
  - Periodically
  - After an insertion or removal of a kernel module
- **Self defense mechanism**
  - By setting hardware breakpoints in crucial parts of the code
    - Just like the black debug register rootkits
- **Hides itself from the kernel, so it cannot be removed**
  - Just like the black rootkits
- **Dumps the code of the malware**
  - Useful for forensics

CERN IT Department
CH-1211 Genève 23
Switzerland
**www.cern.ch/it**

```
[root@kmod-testing sys_call_hijacker]# insmod ./hijacker.ko
[root@kmod-testing sys_call_hijacker]#
Message from syslogd@kmod-testing at Jul 11 14:26:00 ...
 kernel:redeemer: ALERT! sys_call_table IS DIFFERENT - its hash is [0xe6d0d83] and should be [0x65723312]

Message from syslogd@kmod-testing at Jul 11 14:26:00 ...
_kernel:redeemer: Changes detected after loading of hijacker!
```

```
[root@kmod-testing sys_call_hijacker]# tail /var/log/messages -n 11
Jul 11 14:25:47 kmod-testing kernel: redeemer: Rootkit Detector by CERN Security Team
Jul 11 14:25:47 kmod-testing kernel: redeemer:  [+] Logs will be sent at 128.142.172.17
Jul 11 14:25:47 kmod-testing kernel: redeemer:  [+] Deployed
Jul 11 14:26:00 kmod-testing kernel: redeemer: System is clean
Jul 11 14:26:00 kmod-testing kernel: sys_call_hijacker: Injected sys_open
Jul 11 14:26:00 kmod-testing kernel: redeemer: ALERT! sys_call_table IS DIFFERENT - its hash is [0xe6d0d83] and
should be [0x65723312]
Jul 11 14:26:00 kmod-testing kernel: redeemer: Changes detected after loading of hijacker!
Jul 11 14:26:00 kmod-testing kernel: redeemer: Dump of hijacker's init:
Jul 11 14:26:00 kmod-testing kernel: 5531f648c7c7205002a04889e5e81e4c08e148984885c04889058274ffff751448c7c7f0500
2a0e8b6f14ce183c8ffe98000000048c7c70c5202a0ffd04885c04889055174ffff751148c7c7305102a0e88df14ce183c8ffeb5a488b401
04889052b74ffffe886a700e166904889c74881e7fffffeffe885a700e16690488b051474ffff48c74010005002a0e85fa700e166904889c
74881cf00000100e85ea700e1669048c7c7685102a031c0e830f14ce131c0c9c3000000000005500000000000000000000000000000000
0000000000072000100ac5002a0ffffffff000000000000000000000074000200005002a0ffffffff000000000000000000000074000
00400505002a0ffffffff000000000000000000000007400060000e002a0ffffffff000000000000000000000072000800d05002a0fff
fffff000000000000000000000072000900905102a0ffffffff000000000000000000000072000b000c5202a0ffffffff00000000000
0000000000003f000c004083c31500c9ffff000000000000000000000072000d00305202a0ffffffff000000000000000000000072000
00f00405202a0ffffffff000000000000000000000003f001000a085c31500c9ffff000000000000000000000006400110044 5202a0fff
fffff000000000
Jul 11 14:26:00 kmod-testing kernel: redeemer: Dump of hijacker's core:
Jul 11 14:26:00 kmod-testing kernel: 554889e50f1f440000ff1581040000c9c36666666666662e0f1f8400000000000554889e50f1
f4400004889f74889ca48c7c61b5202a0b915000000f3a6c9400f97c60f92c131c04038ce0f44c2c39090554889e5e8675dfee048c7c7d05
002a031c0e87b814de1e8843701e166904889c74881e7fffffeffe8833701e16690488b150a040000488b050b04000048895010e85a3701e
166904889c74881cf00000100e8593701e16690c9c3000400000014000000030000000474e550025e9dc00d52da72a69710c8bd38addac792
009f77379735f63616c6c5f68696a61636b65723a20436c65616e696e672075700a007379735f63616c6c5f68696a61636b65723a206b616
c6c73796d735f6c6f6f6b75705f6e616d652073796d626f6c206e6f7420666f756e640a0000000000000007379735f63616c6c5f68696a616
36b65723a207379735f63616c6c5f7461626c652073796d626f6c206e6f7420666f756e640a0000000007379735f63616c6c5f68696a616
36b65723a20496e6a6563746564207379735f6f70656e0a000000675002a0ffffffff0607ff0100000000785002a0ffffffff0707ff01000
00000915002a0ffffffff0607ff0100000000a25002a0ffffffff0707ff010000000065e002a0ffffffff0607ff010000000076e002a0fff
fffff0707ff010
```

# Work of CST within the kernel

- netlog
- tty-kraven
- redeemer
- dresden

CERN IT Department
CH-1211 Genève 23
Switzerland
**www.cern.ch/it**

# dresden

- Blocks incoming modules
- Logs emergency alerts in case of trying to insert a new module or removing an existing one
- Hides itself from the kernel, so it cannot be removed
  - Just like redeemer (and every malware)
- Open source http://cern-cert.github.com/dresden

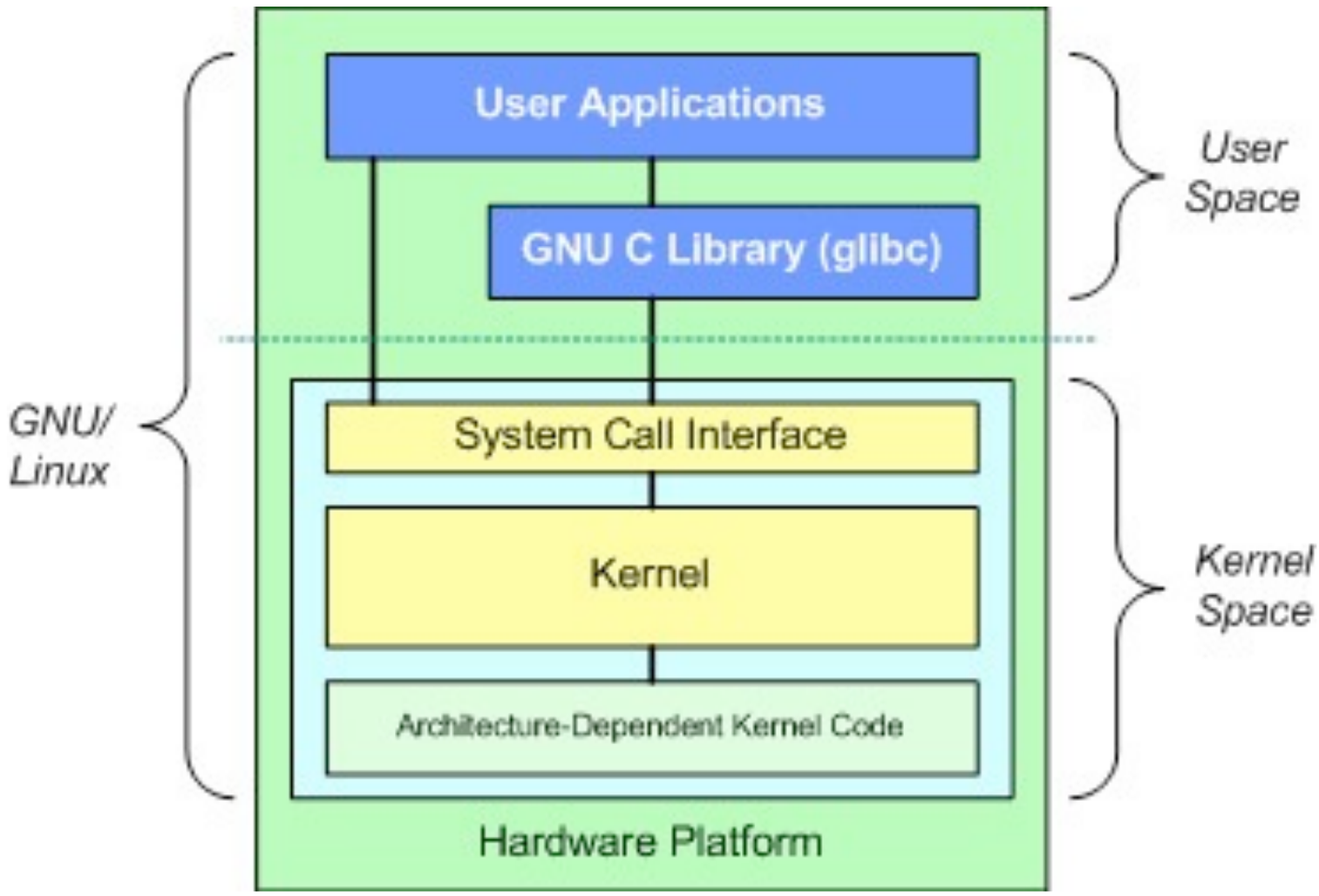
```
[root@kmod-testing sys_call_hijacker]# insmod ./hijacker.ko

Message from syslogd@kmod-testing at Jul 11 14:39:32 ...
 kernel:dresden: event: MODULE_STATE_COMING name: hijacker init: 0xffffffffa0025000 size of init (text + data)
x6e8 core: 0xffffffffa001c000 size of core (text + data) 0x622

Message from syslogd@kmod-testing at Jul 11 14:39:32 ...
 kernel:dresden: New module is th name hijacker. Its functionality will be disabled
insmod: error inserting './hijacker.ko': -1 Operation not permitted
[root@kmod-testing sys_call_hijacker]#
Message from syslogd@kmod-testing at Jul 11 14:39:32 ...
_kernel:dresden: event: MODULE_STATE_GOING name: hijacker core: 0xffffffffa001c000 size of core (text + data) 0x622
```

```
Jul 11 14:39:16 kmod-testing kernel: dresden: Kernel module insertion blocker and action notifier by CERN Security Team
Jul 11 14:39:16 kmod-testing kernel: dresden:   [+] Future loading of kernel modules will be prevented
Jul 11 14:39:16 kmod-testing kernel: dresden:   [+] Emergency messages will be logged in case of trying to load or unload a module
Jul 11 14:39:16 kmod-testing kernel: dresden:   [+] You are not able to remove this module
Jul 11 14:39:32 kmod-testing kernel: dresden: event: MODULE_STATE_COMING name: hijacker init: 0xffffffffa0025000 size of init (text + data)
x6e8 core: 0xffffffffa001c000 size of core (text + data) 0x622
Jul 11 14:39:32 kmod-testing kernel: dresden: New module is th name hijacker. Its functionality will be disabled
Jul 11 14:39:32 kmod-testing kernel: Module insertion blocked by CERN's dresden
Jul 11 14:39:32 kmod-testing kernel: dresden: event: MODULE_STATE_GOING name: hijacker core: 0xffffffffa001c000 size of core (text + data) 0x622
```

# Overview

- Work of CST within the kernel
- **Architecture of an OS**
- Anatomy of a kernel module
- kprobes
- Debugging
- Good and Bad practices

# Architecture of an OS



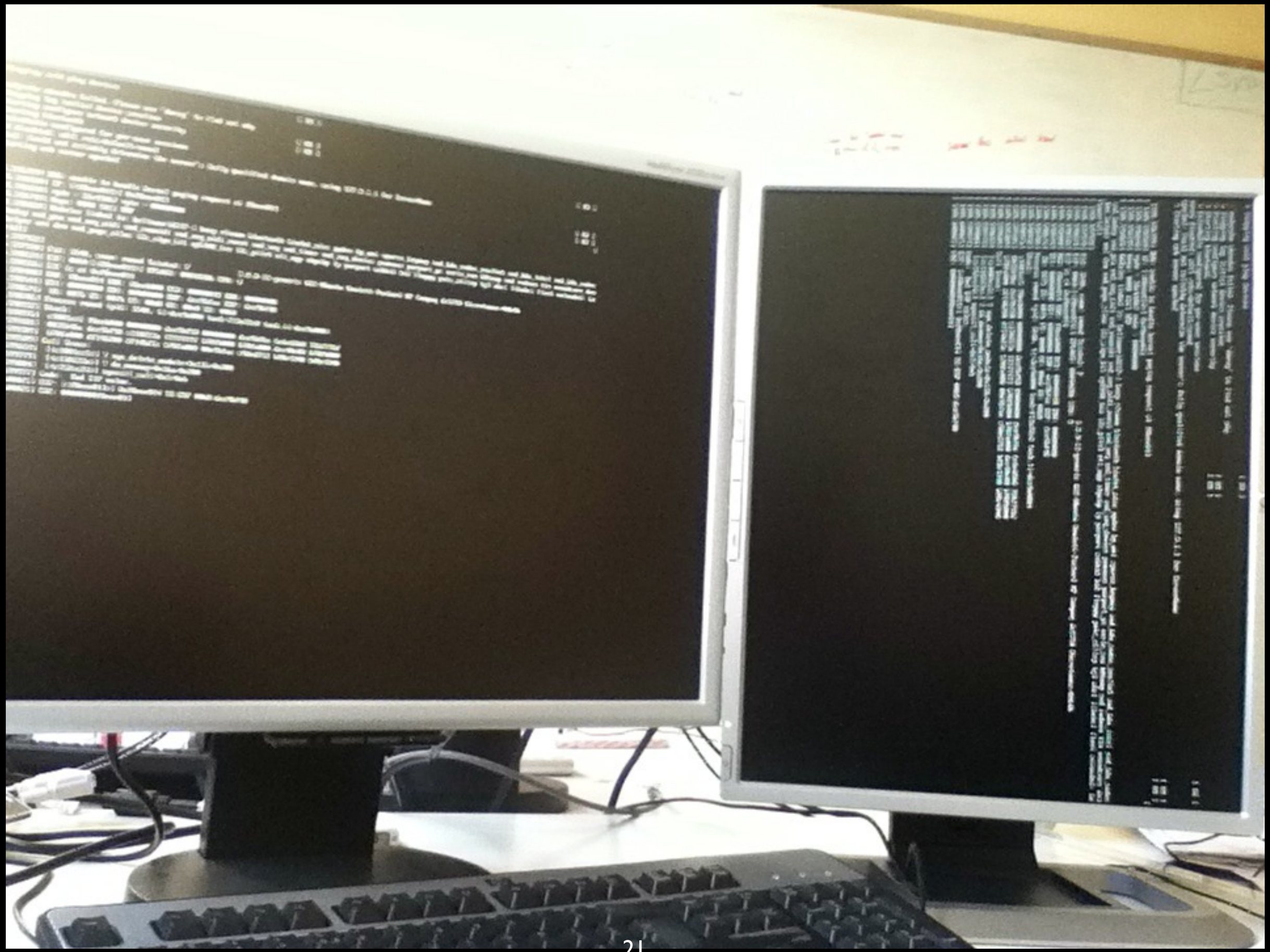

CERN IT Department
CH-1211 Genève 23
Switzerland
**www.cern.ch/it**

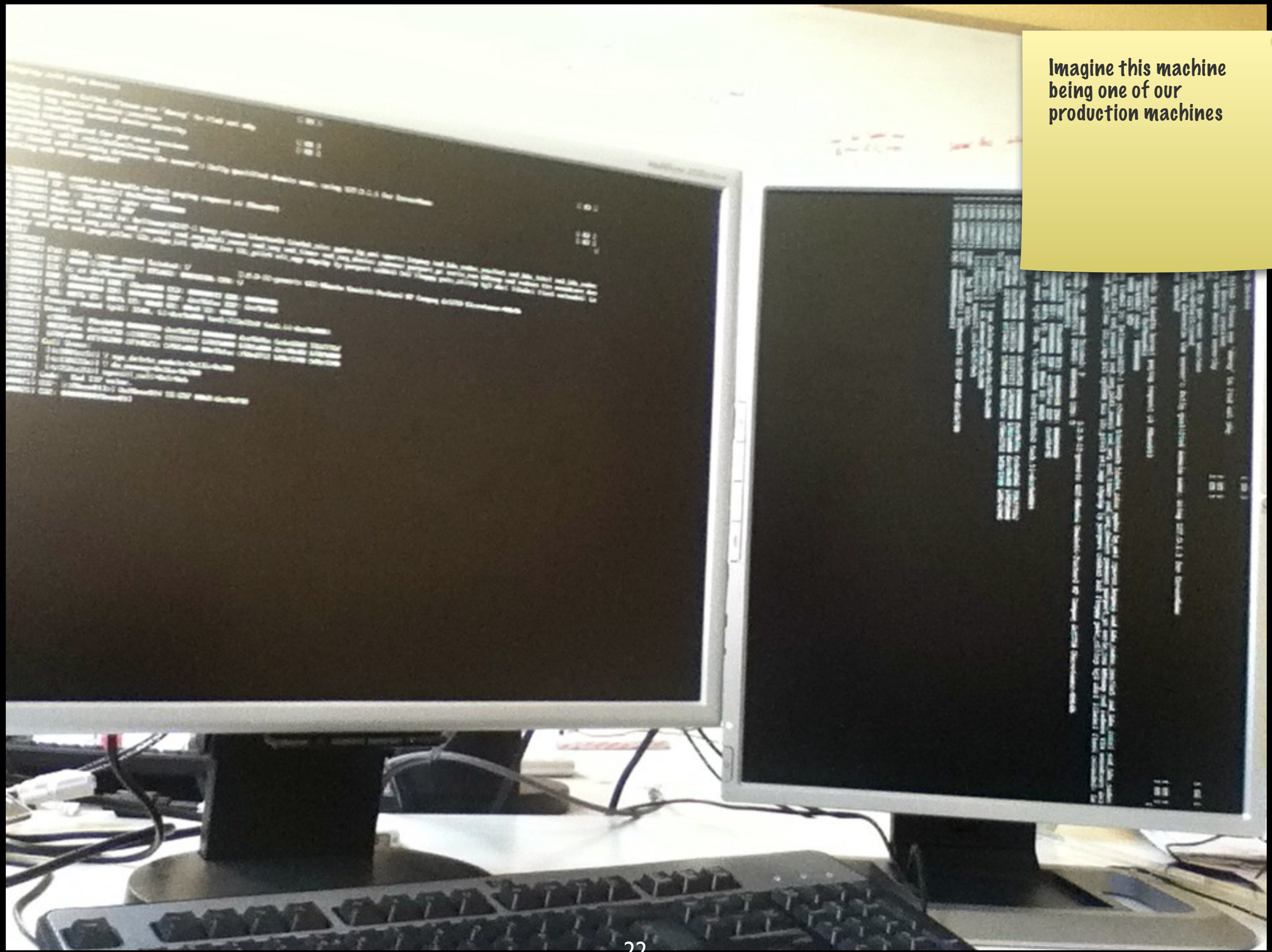
# Architecture of an OS

- **The kernel is**
  - The heart of an OS
  - Serving the requests from user space
    - system calls
    - proc filesystem
    - ...
  - Protecting user space software from errors
    - Segmentation fault
    - Buss error
    - ...

# What can go wrong?

- **Bug in the kernel will cause**
  - System crash
    - with logs and dump of registers (if you are lucky)
  - System hang up
    - no logs at all...
  - Filesystem corruption
    - As bad as it sounds :(





Imagine this machine being one of our production machines


NOT ANOTHER SHALLOW
HOLLYWOOD MOVIE.
DISASTER
MOVIE
AL GORE WAS RIGHT.
IN THEATERS AUGUST 29
LIONSGATE

# Overview

- Work of CST within the kernel
- Architecture of an OS
- **Anatomy of a kernel module**
- kprobes
- Debugging
- Good and Bad practices

# Anatomy of a kernel module

- **What is a kernel module?**
  - Extends (or changes) the behavior of the system's kernel
  - A kernel object file (*.ko) that can be inserted/removed in/from the kernel, on the fly
    - No reboot! :)

# Anatomy of a kernel module

- **How to insert**
  - insmod <kernel object file name> <parameters>
    - or modprobe (more clever tool than insmod)
- **When inserting a module, the init_module function runs**
  - Parameters can be given to the module
- **You need root privileges**

# Anatomy of a kernel module

- How to remove
  - rmmod <kernel object name>
- When removing a module, the cleanup_module function runs
- You need root privileges

# Anatomy of a kernel module

- **Retrieve information about the module**
  - modinfo <kernel object name>
- **Information that the developer of the module wants to export**
  - Module description
  - Author
  - License
  - Parameters
    - Names
    - Description

```
[root@kmod-testing redeemer]# modinfo redeemer
filename:       /lib/modules/2.6.32-279.el6.x86_64/extra/redeemer/redeemer.ko
license:        GPL
description:    Linux rootkit detector
author:         CERN's Security Team (http://security.cern.ch)
srcversion:     A34C6DF1B76EDFB15A939D7
depends:
vermagic:       2.6.32-279.el6.x86_64 SMP mod_unload modversions
parm:           remote_log_ip:IPv4 address for remote syslog server. Defaults to localhost (charp)
parm:           delay_between_log_marks:Interval in seconds between 2 "mark" messages in the log. Defaults to 86400 (1 day) (uint)
parm:           integrity_checker_interval:Interval in milliseconds between 2 integrity checks. Defaults to 15000 (uint)
parm:           log_detailed:Should logging be detailed ? - Defaults to no (int)
```

# Anatomy of a kernel module

- **Messages from modules are not visible in the terminal**
- **klogd fetches the messages and delivers them to syslogd**
  - Messages visible in /var/log/messages
    - tail -f /var/log/messages | grep <module tag>
  - You need root privileges
- **dmesg (raw kernel logs)**
  - No need of root priviledges

CERN IT Department
CH-1211 Genève 23
Switzerland
**www.cern.ch/it**

# SENSE

This picture makes none

```c
#include <linux/module.h>

#define MODULE_NAME "hello_world"

int init_module(void)
{

  printk(KERN_INFO MODULE_NAME "Hello world!\n");
  return 0;
}
```

```c
void cleanup_module(void)
{
  printk(KERN_INFO MODULE_NAME "Goodbye world\n");
}

MODULE_DESCRIPTION("Hello world LKM\n");
MODULE_AUTHOR("Panos Sakkos <panos.sakkos@cern.ch>");
MODULE_LICENSE("GPL");
```

```
[jeni] /afs/cern.ch/user/p/psakkos > insmod ./hello_world.ko
[jeni] /afs/cern.ch/user/p/psakkos > tail /var/log/messages -n 1
May 31 15:33:56 jeni kernel: hello_world: Hello world!
[jeni] /afs/cern.ch/user/p/psakkos > rmmod hello_world
[jeni] /afs/cern.ch/user/p/psakkos > tail /var/log/messages -n 1
May 31 15:34:17 jeni kernel: hello_world: Goodbye world
```

# Overview

- Work of CST within the kernel
- Architecture of an OS
- Anatomy of a kernel module
- **kprobes**
- Debugging
- Good and Bad practices

# kprobes

- **Plant probes in (almost) everywhere within the kernel space memory**
  - Ideal for monitoring
    - netlog
  - Useful for enhancing security of the kernel
    - tty-kraven
  - But also for attacking...
    - Malwares that use handlers that execute before **and** after the execution of the probed memory address

CERN IT Department
CH-1211 Genève 23
Switzerland
**www.cern.ch/it**

```c
#include <linux/module.h>
#include <linux/kprobes.h>

#define MODULE_NAME "dummy_probes"
#define EREGISTER 1

static int pre_handler_sys_read(struct kprobe *p,
                                struct pt_regs *regs){
  printk(KERN_INFO MODULE_NAME ": sys_read called\n");
  return 0;
}
```

```
static struct kprobe kprobe =
{
        .pre_handler = pre_handler_sys_read,
        .post_handler = NULL,
        .fault_handler = NULL,
        .symbol_name = "sys_read",
};
```

```c
int init_module(void)
{
  if(register_kprobe(&kprobe) < 0) {
    printk(KERN_ERR MODULE_NAME ": Failed to plant probe\n");
    return -EREGISTER;
  }
  printk(KERN_INFO MODULE_NAME ": Probe planted\n");
  return 0;
}
```

```
void cleanup_module(void){
  unregister_kprobe(&kprobe);
  printk(KERN_INFO MODULE_NAME ": Probe unplanted\n");
}

MODULE_DESCRIPTION("A demo example on using kprobes API\n");
MODULE_AUTHOR("Panos Sakkos <panos.sakkos@cern.ch>");
MODULE_LICENSE("GPL");
```

```
May 31 16:33:10 jeni kernel: dummy_probes: Probe planted
May 31 16:33:10 jeni kernel: dummy_probes: sys_read called
May 31 16:33:12 jeni last message repeated 50581 times
May 31 16:33:12 jeni kernel: dummy_probes: Probe unplanted
```

# Overview

- Work of CST within the kernel
- Architecture of an OS
- Anatomy of a kernel module
- kprobes
- **Debugging**
- Good and Bad practices

# Debugging

- **gdbmod**
  - Painful to setup
  - Offers single stepping
- **User Mode Linux**
  - Debug the whole Operating System as a process => gdb => single stepping
  - Easy to setup
  - http://user-mode-linux.sourceforge.net/

CERN IT Department
CH-1211 Genève 23
Switzerland
**www.cern.ch/it**

# Overview

- Work of CST within the kernel
- Architecture of an OS
- Anatomy of a kernel module
- kprobes
- Debugging
- **Good and Bad practices**

# Do not develop a kernel module if

- You are not a C guru
- You are not familiar with OSs
- You don’t have previous experience with an assembly language
- You don’t have patience...

# Why do I need to be patient?

- **Sometimes the kernel documentation is (extremely) poor or even it doesn't exist**
  - You need to read the kernel source in order to see how to use certain structures and APIs

# Why do I need to be patient?

- **Painful and time consuming development**
  - Locking/Unlocking on every resource that you use
  - Difficult to debug
    - logs may not be flushed before crashing of the machine
      - no logs :(
  - Develop code that targets different kernel versions
  - Crash of your module means crash of the system...
    - Reboot development (virtual) machine

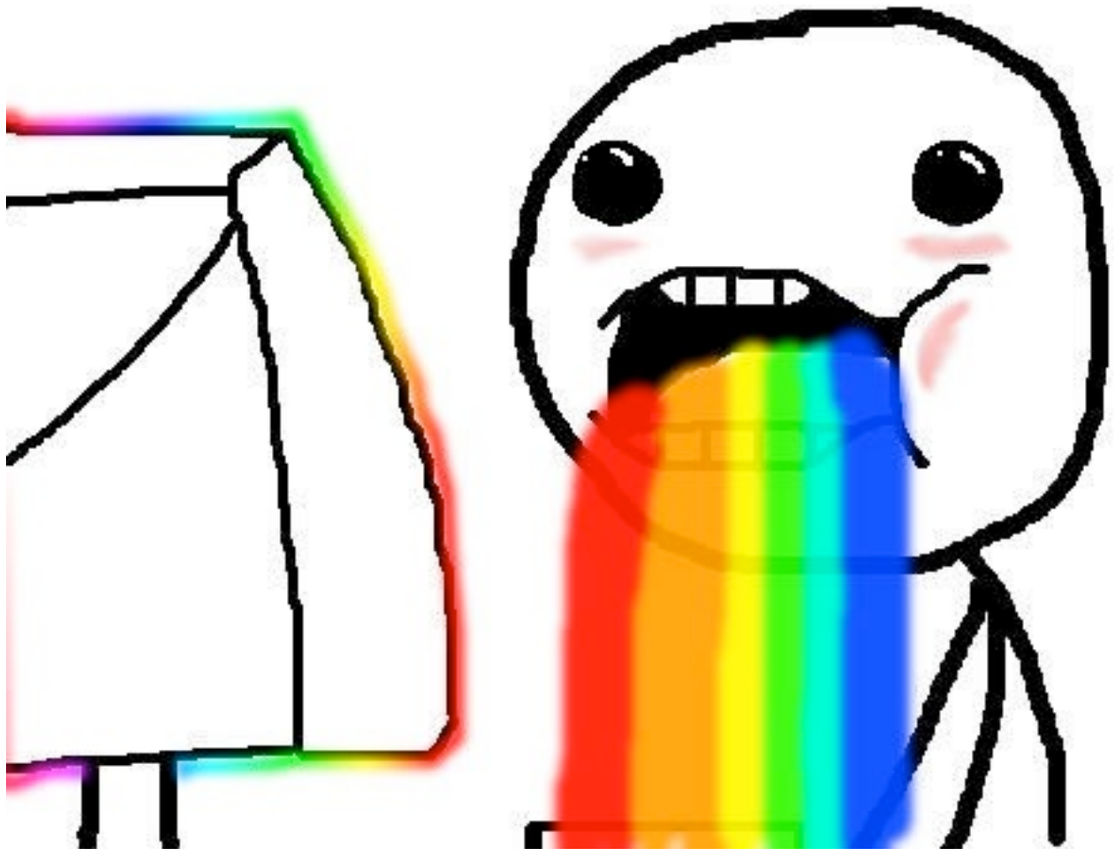
# When to develop a kernel module

- Access to resources that are not available in user space
- System monitoring
- Enhance defense of a system
- Attack a system...
- Implement new system calls
- Drivers
- ...

# Kernel space visible resources

- **Literally <u>everything</u>**
  - The whole system's memory
  - Registers
    - control registers
    - debug registers
    - ...
  - Access to hardware
  - Scheduling structures
  - Process structures
  - Thread structures
  - Interrupt table/handlers
  - system call table
  -

# Code practices



CERN IT Department
CH-1211 Genève 23
Switzerland
**www.cern.ch/it**

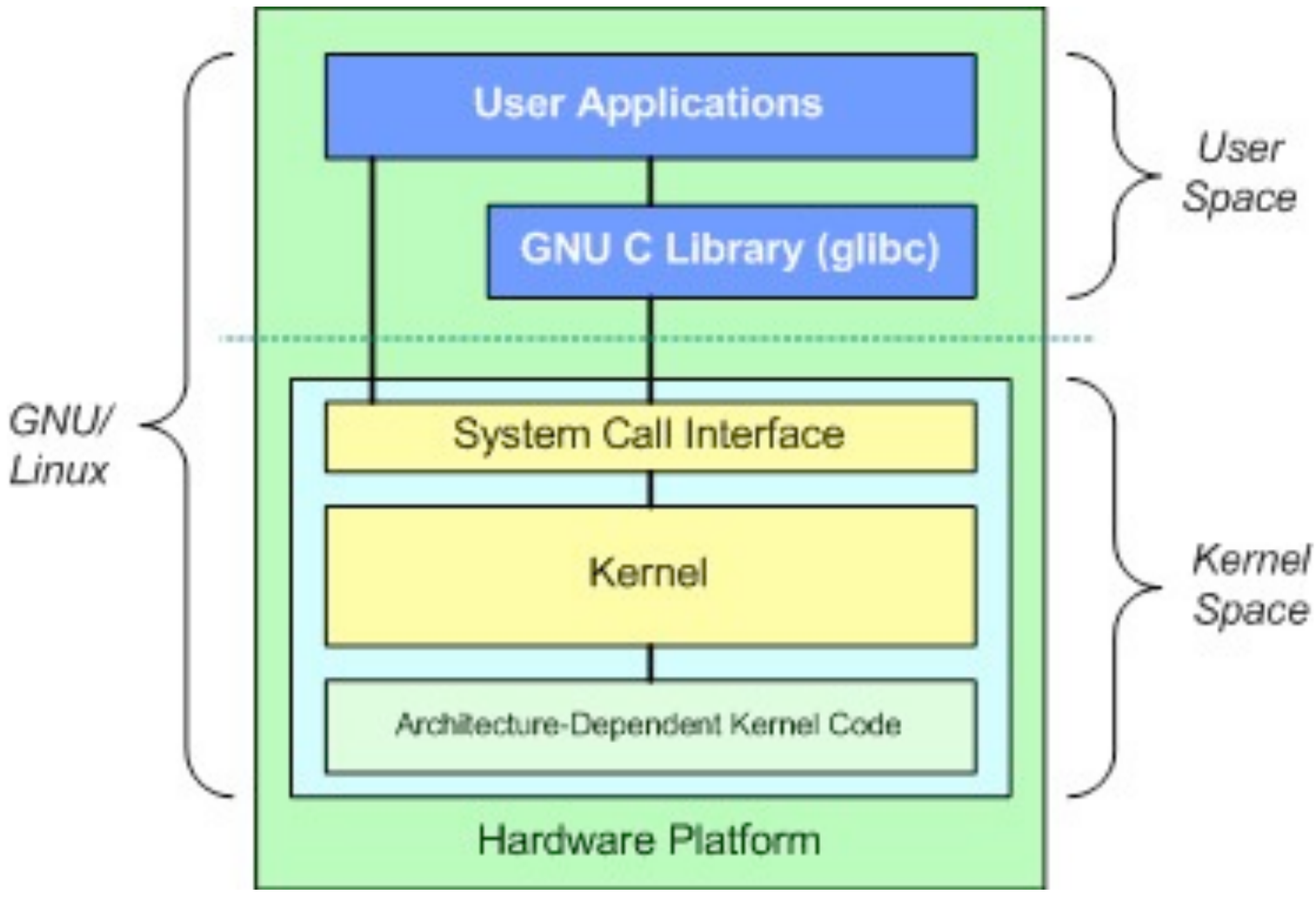
# Calling system calls

CERN IT Department

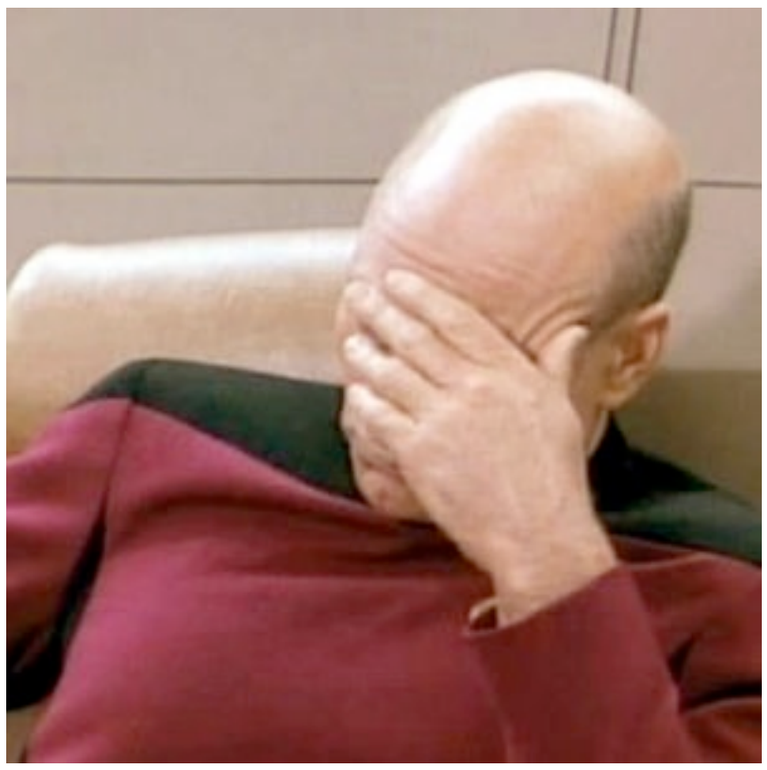
?

CERN IT Department
CH-1211 Genève 23
Switzerland
**www.cern.ch/it**

# Calling system calls



# Calling system calls...



# File I/Os

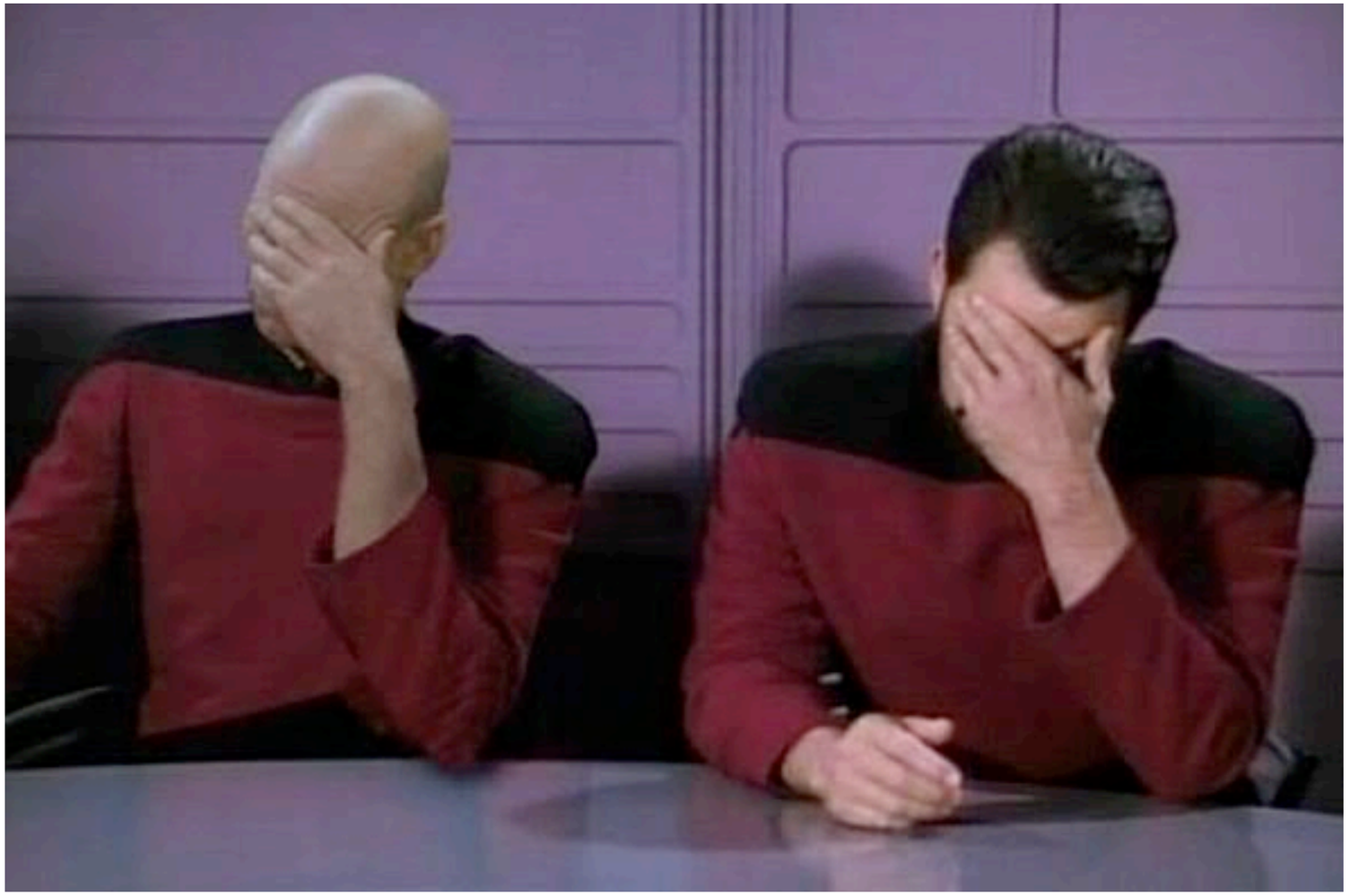
?

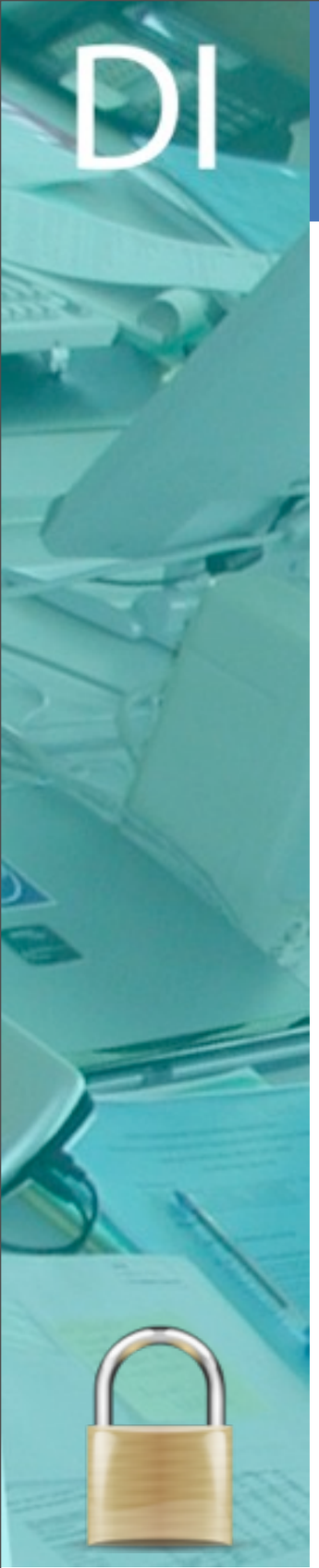
# File I/Os

- open, read and write functions are system calls

# File I/Os...







CERN IT Department
CH-1211 Genève 23
Switzerland
**www.cern.ch/it**

# Communication with user space?

# Communication with user space

- **proc file system**
- **Module parameters**
  - While inserting the module in the kernel
- **...**

# Use as little assembly as you can

- Find APIs that encapsulate the assembly code that you want to use
- The person that will inherit your work will love you :)

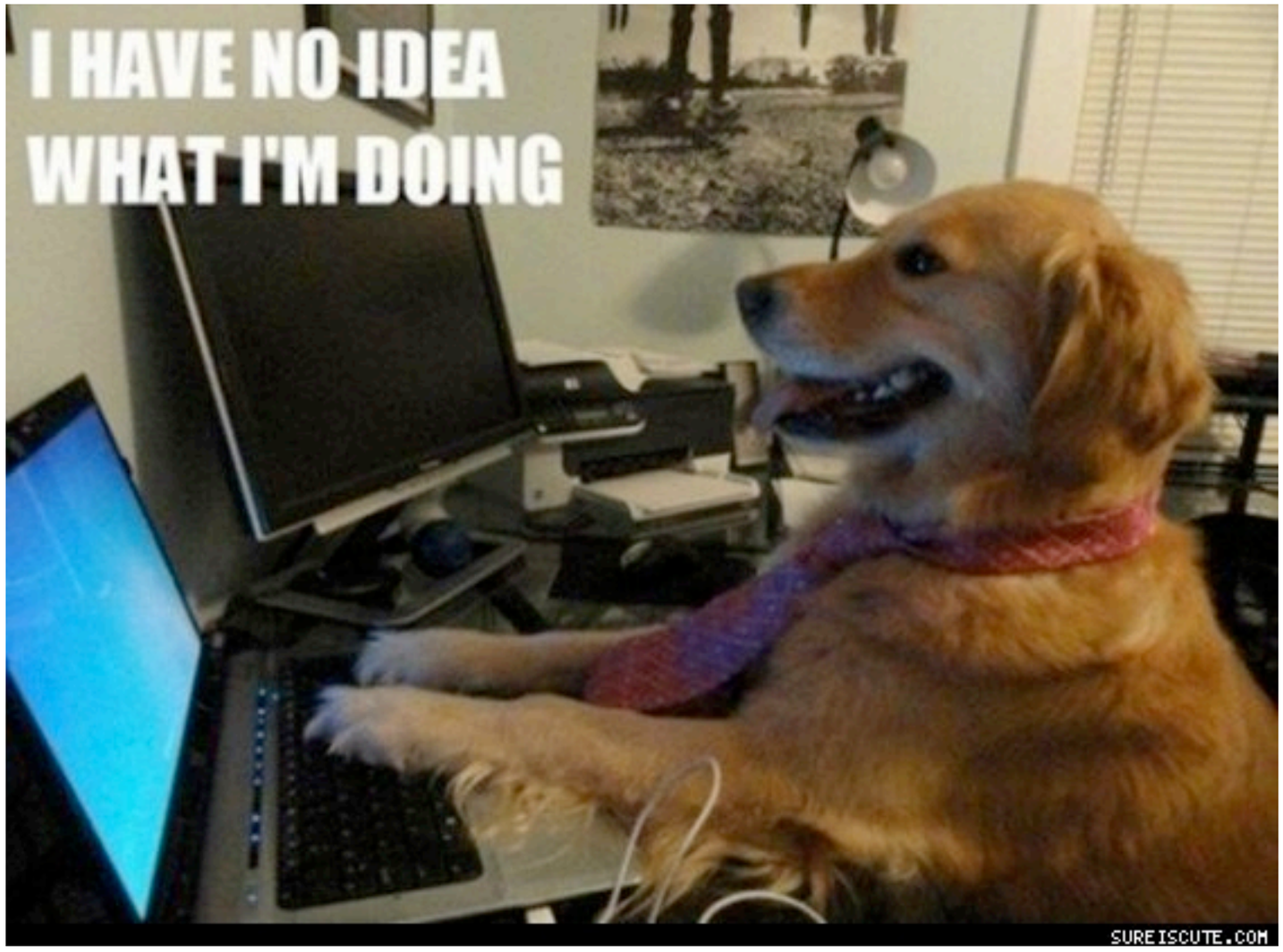
# Use as little assembly as you can

- **Some months ago we inherited this code:**

```
#ifdef CONFIG_X86_64
      __asm__ __volatile__ ("movq %0,%%dr0\n" :: "r" (arg->p_dr0));
#else
      __asm__ __volatile__ ("movl %0,%%dr0\n" :: "r" (arg->p_dr0));
#endif
   if (arg->p_dr1)
#ifdef CONFIG_X86_64
      __asm__ __volatile__ ("movq %0,%%dr1\n" :: "r" (arg->p_dr1));
#else
      __asm__ __volatile__ ("movl %0,%%dr1\n" :: "r" (arg->p_dr1));
#endif
   if (arg->p_dr2)
#ifdef CONFIG_X86_64
      __asm__ __volatile__ ("movq %0,%%dr2\n" :: "r" (arg->p_dr2));
#else
      __asm__ __volatile__ ("movl %0,%%dr2\n" :: "r" (arg->p_dr2));
#endif
   if (arg->p_dr3)
#ifdef CONFIG_X86_64
      __asm__ __volatile__ ("movq %0,%%dr3\n" :: "r" (arg->p_dr3));
#else
      __asm__ __volatile__ ("movl %0,%%dr3\n" :: "r" (arg->p_dr3));
#endif

   /* Status... */
   if (arg->p_stat) {
#ifdef CONFIG_X86_64
      p_tmp = arg->p_stat & 0x00000000FFFFFFFF;
      __asm__ __volatile__ ("mov %0,%%dr6\n" :: "r" (p_tmp));
#else
      __asm__ __volatile__ ("movl %0,%%dr6\n" :: "r" (arg->p_stat));
#endif
   }
   /* Control... */
   if (arg->p_ctrl) {
#ifdef CONFIG_X86_64
      p_tmp = arg->p_ctrl & 0x00000000FFFFFFFF;
      __asm__ __volatile__ ("mov %0,%%dr7\n" :: "r" (p_tmp));
#else
      __asm__ __volatile__ ("movl %0,%%dr7\n" :: "r" (arg->p_ctrl));
#endif
```

CERN IT Department
CH-1211 Genève 23
Switzerland
**www.cern.ch/it**

# Use as little assembly as you can



# Use as little assembly as you can

- We refactored it into this:

```
set_debugreg(arg->p_dr0, 0);
set_debugreg(arg->p_dr1, 1);
set_debugreg(arg->p_dr2, 2);
set_debugreg(arg->p_dr3, 3);
/* reset condition */
set_debugreg(0, 6);
/* reset control */
set_debugreg(CONTROL_REGISTER, 7);
```

CERN IT Department
CH-1211 Genève 23
Switzerland
**www.cern.ch/it**

# Supporting Multiple Versions

- Use macros that compile different code for each kernel version

```
#if LINUX_VERSION_CODE < KERNEL_VERSION(2, 6, 29)
        #define get_current_uid() current->uid
#else
        #define get_current_uid() current_uid()
#endif
```

# Testing

- **Test your module**
  - Against all the major kernel versions that is designed to support
  - Against every architecture that you support
    - Even if you don't use explicitly assembly code
      - Functions that you call in your code do
        - We had a bug once from a kernel call that used assembly code...
  - Against real **and** Virtual Machines

CERN IT Department
CH-1211 Genève 23
Switzerland
**www.cern.ch/it**

# Supporting Multiple Versions

- **How can I detect changes that affect my module?**
  - Compile against every kernel version that you support
- **How can I find what the change was?**
  - Download the source code of the problematic version
    - grep
  - Search on a linux reference site
    - searches in the source of desired kernel version
    - i.e. http://lxr.linux.no
- **CST develops kernel modules that run from 2.6.18 (SLC 5) up to 3.3.8 (almost the latest) linux kernel :)**

# OOP in the kernel

- *"No, God no."*
  - from http://www.kernelnewbies.org/ FAQ
- **Performance is crucial**
- **Legacy...**

# Assertions

- **BUG_ON(undesired condition)**
  - macro
  - Dumps registers
  - Crashes the system
  - Used for serious problems
- **WARN_ON(undesired condition)**
  - macro
  - Dumps registers
  - Continues execution
  - Used for recoverable problems

# Boost performance

- **likely/unlikely macros**
  - Used in conditions
  - 1 or 2 less jump assembly commands
    - Order of magnitude: nanoseconds
  - Used in probes of netlog
    - Called tenths of thousands per second
- **Initialize variables only where needed**
- **As little as possible amount of code that has disabled interrupts or scheduling**
- **Define register variables**
- **...**

# Boost performance

```
if(unlikely(!is_tcp(sock)) || unlikely(!is_inet(sock)))
{
        goto out;
}

if(unlikely(current == NULL))
{
        goto out;
}
```

# Overview

- Work of CST within the kernel
- Architecture of an OS
- Anatomy of a kernel module
- kprobes
- Debugging
- Good and Bad practices

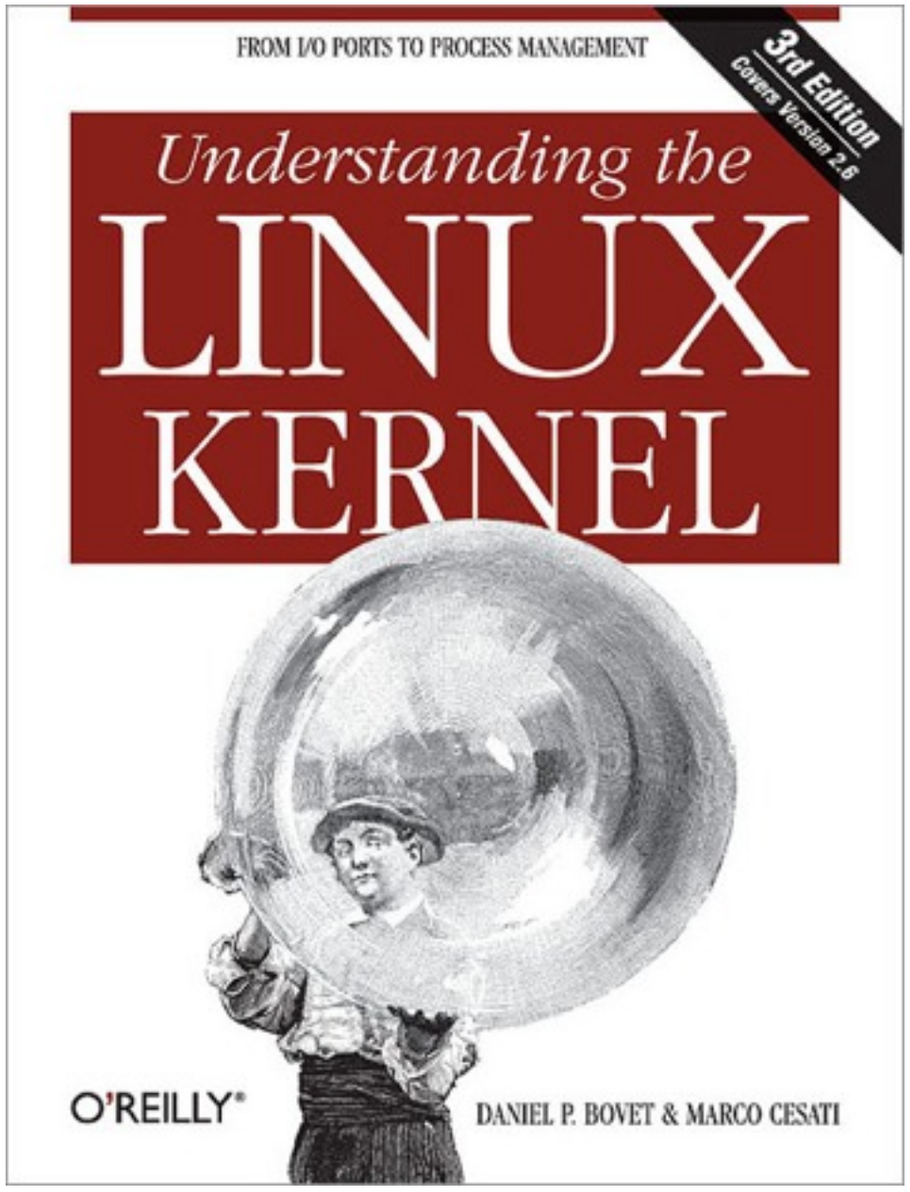
CERN IT Department
CH-1211 Genève 23
Switzerland
**www.cern.ch/it**

# Useful links

- http://www.kernel.org - kernel source
- http://lxr.linux.no - linux kernel reference site
- http://www.kernelnewbies.org/ - community of kernel hackers
- http://linuxjournal.com - articles about the kernel

# Books

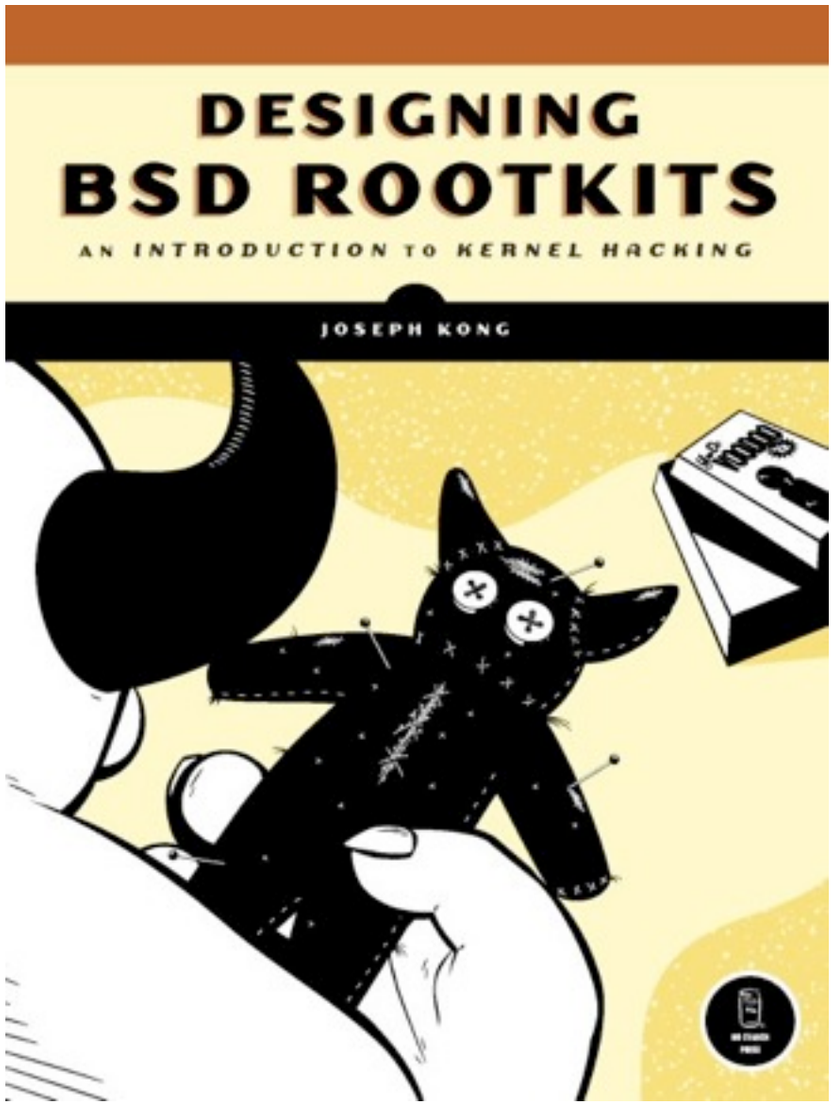





CERN IT Department
CH-1211 Genève 23
Switzerland
**www.cern.ch/it**

# Books



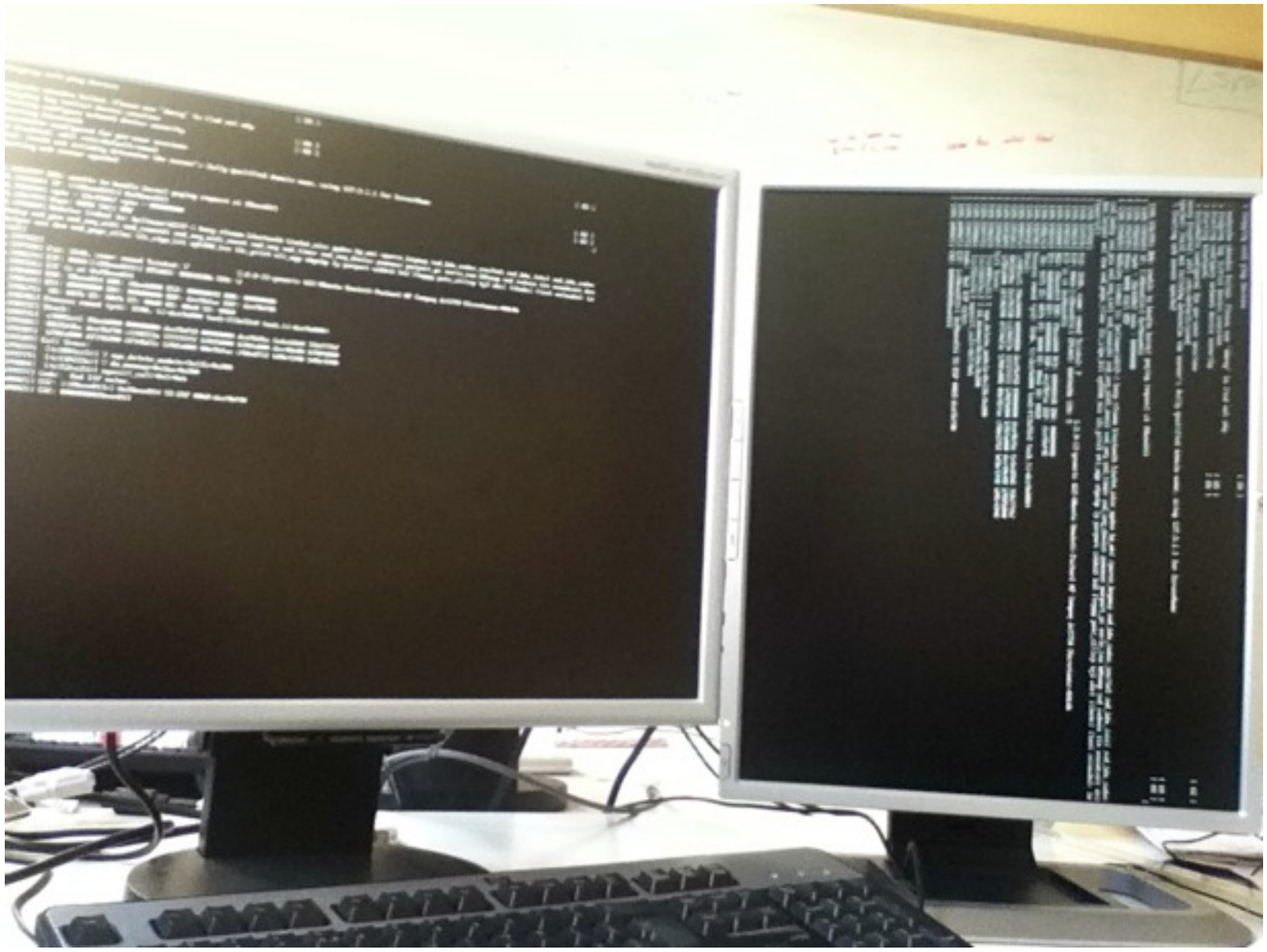



CERN IT Department
CH-1211 Genève 23
Switzerland
**www.cern.ch/it**

# Questions?

CERN IT Department
CH-1211 Genève 23
Switzerland
**www.cern.ch/it**

# Thank you



CERN IT Department
CH-1211 Genève 23
Switzerland
**www.cern.ch/it**