

Linux malware development 1: Intro to kernel hacking. Simple C example.

 cocomelonc.github.io/linux/2024/06/20/linux-kernel-hacking-1.html

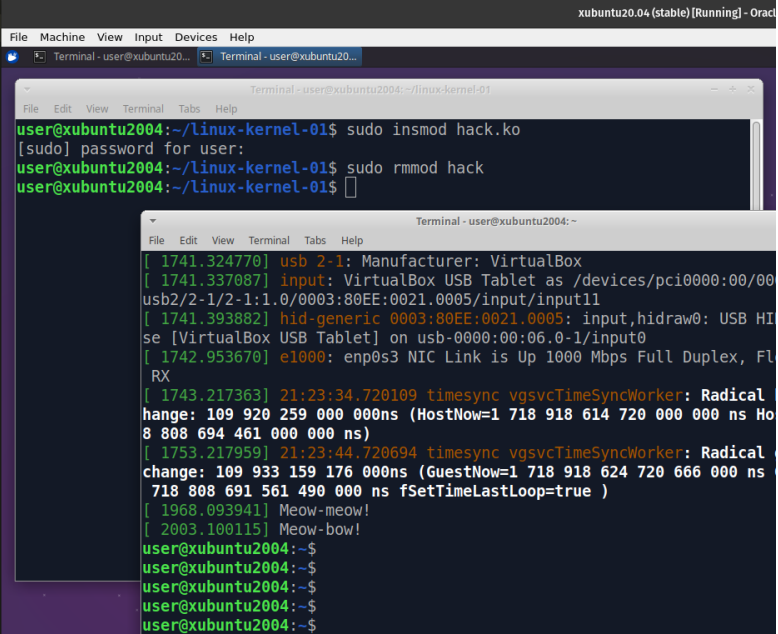
June 20, 2024



5 minute read

Hello, cybersecurity enthusiasts and white hackers!

```
024-06-20-linux-kernel-hacking-1 > C hack.c
7 #include <linux/init.h>
8 #include <linux/module.h>
9 #include <linux/kernel.h>
10
11 MODULE_LICENSE("GPL");
12 MODULE_AUTHOR("cocomelon");
13 MODULE_DESCRIPTION("kernel-test-01");
14 MODULE_VERSION("0.001");
15
16 static int __init hack_init(void) {
17     printk(KERN_INFO "Meow-meow!\n");
18     return 0;
19 }
20
21 static void __exit hack_exit(void) {
22     printk(KERN_INFO "Meow-bow!\n");
23 }
24
25 module_init(hack_init);
26 module_exit(hack_exit);
27
```



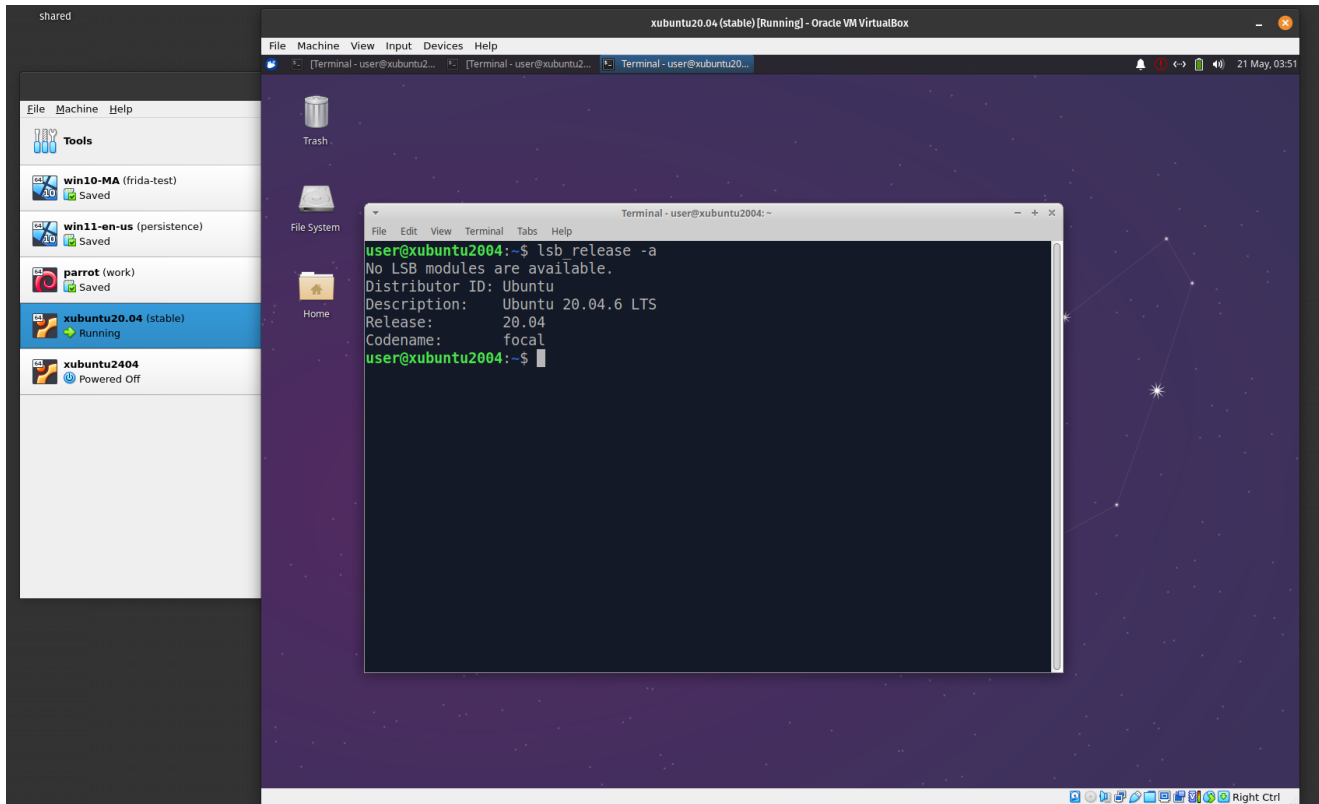
```
xubuntu20.04 (stable) [Running] - Oracle VM
Terminal - user@xubuntu20...
Terminal - user@xubuntu20...
Terminal - user@xubuntu2004: ~/linux-kernel-01
user@xubuntu2004:~/linux-kernel-01$ sudo insmod hack.ko
[sudo] password for user:
user@xubuntu2004:~/linux-kernel-01$ sudo rmmod hack
user@xubuntu2004:~/linux-kernel-01$
Terminal - user@xubuntu2004:~
File Edit View Terminal Tabs Help
[ 1741.324770] usb 2-1: Manufacturer: VirtualBox
[ 1741.337087] input: VirtualBox USB Tablet as /devices/pci0000:00/0000:02:00.0/usb2/2-1/2-1:1.0/0003:80EE:0021.0005/input/input11
[ 1741.393882] hid-generic 0003:80EE:0021.0005: input,hidraw0: USB HID v1.10 Generic Device [VirtualBox USB Tablet] on usb-0000:00:06.0-l/input0
[ 1742.953670] e1000: enp0s3 NIC Link is Up 1000 Mbps Full Duplex, Flow Control: RX
[ 1743.217363] 21:23:34.720109 timesync vgsvcTimeSyncWorker: Radical change: 109 920 259 000 000ns (HostNow=1 718 918 614 720 000 000 ns HostNow=8 808 694 461 000 000 ns)
[ 1753.217959] 21:23:44.720694 timesync vgsvcTimeSyncWorker: Radical change: 109 933 159 176 000ns (GuestNow=1 718 918 624 720 666 000 ns HostNow=718 808 691 561 490 000 ns fSetTimeLastLoop=true)
[ 1968.093941] Meow-meow!
[ 2003.100115] Meow-bow!
user@xubuntu2004:~$
user@xubuntu2004:~$
user@xubuntu2004:~$
user@xubuntu2004:~$
user@xubuntu2004:~$
```

In fact, this post could be called something else like “*Malware development trick part 41*”, but here I again answer many questions that my readers ask me. *How can I develop malware for linux?*

Perhaps this post will be the beginning and also the starting point for a series of posts (those who have been reading me for a long time have probably noticed that I have many different series of posts that I started but have not yet brought these series to their logical end).

To be honest, my last experience of programming for Linux kernel was at the university about 10+ years ago, since then a lot has changed, so I decided to try to write something interesting like malware: linux rootkit, stealer, etc....

First of all, I installed a linux virtual machine - xubuntu 20.04 so as not to break anything in my system. I think you can install a more recent version of Ubuntu (Xubuntu, Lubuntu), but version 20.04 is quite suitable for experiments:



practical example

For example if we need create a malware, like a kernel rootkit, the code we develop will have the ability to execute with kernel level privileges (**ring 0**) using the kernel modules we create. Working in this role can have its challenges. On one hand, our work goes unnoticed by the user and userspace tools. However, if we make a mistake, it can have serious consequences. The kernel is unable to protect us from its own flaws, which means we risk crashing the entire system. Using VM will help alleviate the challenges of developing in our xubuntu, making it a much more manageable requirement.

Let's start from import modules:

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
```

These **#include** statements include the necessary header files for kernel module programming:

- **linux/init.h** - contains macros and functions for module initialization and cleanup.
- **linux/module.h** - contains macros and functions for module programming.
- **linux/kernel.h** - provides various functions and macros for kernel development.

```
MODULE_LICENSE("GPL");
MODULE_AUTHOR("cocomelonc");
MODULE_DESCRIPTION("kernel-test-01");
MODULE_VERSION("0.001");
```

These macros define metadata about the module:

- `MODULE_LICENSE("GPL")` - specifies the license under which the module is released. Here, it's the GNU General Public License.
- `MODULE_AUTHOR("cocomelonc")` - specifies the author of the module.
- `MODULE_DESCRIPTION("kernel-test-01")` - Provides a description of the module.
- `MODULE_VERSION("0.001")` - specifies the version of the module.

At the next few lines we are define initialization function:

```
static int __init hack_init(void) {
    printk(KERN_INFO "Meow-meow!\n");
    return 0;
}
```

This function is the initialization function for the module:

- `static int __init hack_init(void)` - defines the function as a static function (local to this file) and marks it as an initialization function using the `__init` macro.
- `printk(KERN_INFO "Meow-meow!\n")` - prints the message "Meow-meow!" to the kernel log with an informational log level.
- `return 0` - returns 0 to indicate successful initialization.

Next one is the `hack_exit` function:

```
static void __exit hack_exit(void) {
    printk(KERN_INFO "Meow-bow!\n");
}
```

This function is the cleanup function for the module:

- `static void __exit hack_exit(void)` - defines the function as a static function and marks it as an exit (cleanup) function using the `__exit` macro.
- `printk(KERN_INFO "Meow-bow!\n")` - prints the message "Meow-bow!" to the kernel log with an informational log level.

Then, registering the initialization and cleanup functions:

```
module_init(hack_init);
module_exit(hack_exit);
```

So, the full source code is looks like this `hack.c`:

```

/*
 * hack.c
 * introduction to linux kernel hacking
 * author @cocomelonc
 * https://cocomelonc.github.io/linux/2024/06/20/kernel-hacking-1.html
 */
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>

MODULE_LICENSE("GPL");
MODULE_AUTHOR("cocomelonc");
MODULE_DESCRIPTION("kernel-test-01");
MODULE_VERSION("0.001");

static int __init hack_init(void) {
    printk(KERN_INFO "Meow-meow!\n");
    return 0;
}

static void __exit hack_exit(void) {
    printk(KERN_INFO "Meow-bow!\n");
}

module_init(hack_init);
module_exit(hack_exit);

```

This code demonstrates the basic structure of a Linux kernel module, including how to define initialization and cleanup functions and how to provide metadata about the module.

demo

Let's go to see this module in action. Before compiling you need install:

```

$ apt update
$ apt install build-essential linux-headers-$(uname -r)

```

For compiling create **Makefile** file with the following content:

```

obj-m += hack.o

all:
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

clean:
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean

```

The provided **Makefile** is used to compile and clean a Linux kernel module. **obj-m** variable is used to list the object files to be built as kernel modules. **hack.o** is the object file that will be built from the **hack.c** source file. The **+=** operator adds **hack.o** to the list of object files to be compiled as modules.

```
make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
```

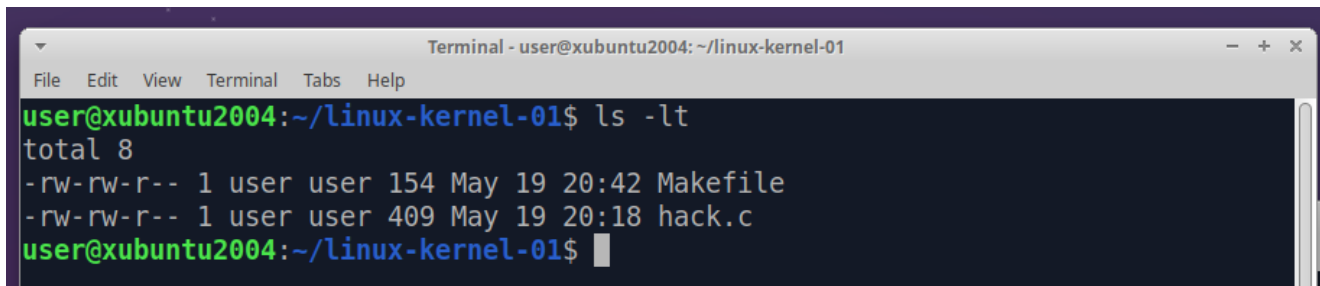
This command invokes `make` to compile the module. `-C /lib/modules/$(shell uname -r)/build` changes the directory to the build directory of the currently running kernel. `$(shell uname -r)` gets the version of the currently running kernel, and `/lib/modules/$(shell uname -r)/build` is where the kernel build directory is located.

`M=$(PWD)` sets the `M` variable to the current working directory `$(PWD)`, which is where your module source code is located. This tells the kernel build system to look in the current directory for the module source files.

and `modules` this target in the kernel build system compiles the modules listed in `obj-m`.

`make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean` - this command cleans up the module build files.

Open a terminal, navigate to the directory containing `hack.c` and `Makefile`:

A terminal window titled "Terminal - user@xubuntu2004: ~/linux-kernel-01" with a menu bar (File, Edit, View, Terminal, Tabs, Help). The terminal shows the command `ls -lt` and its output:

```
total 8
-rw-rw-r-- 1 user user 154 May 19 20:42 Makefile
-rw-rw-r-- 1 user user 409 May 19 20:18 hack.c
user@xubuntu2004:~/linux-kernel-01$
```

and run the following command to compile the module:

```
make
```

```
Terminal - user@xubuntu2004: ~/linux-kernel-01
File Edit View Terminal Tabs Help
user@xubuntu2004:~/linux-kernel-01$ make
make -C /lib/modules/5.15.0-107-generic/build M=/home/user/linux-kernel-01 modules
make[1]: Entering directory '/usr/src/linux-headers-5.15.0-107-generic'
  CC [M] /home/user/linux-kernel-01/hack.o
  MODPOST /home/user/linux-kernel-01/Module.symvers
  CC [M] /home/user/linux-kernel-01/hack.mod.o
  LD [M] /home/user/linux-kernel-01/hack.ko
  BTF [M] /home/user/linux-kernel-01/hack.ko
Skipping BTF generation for /home/user/linux-kernel-01/hack.ko due to unavailability of vmlinux
make[1]: Leaving directory '/usr/src/linux-headers-5.15.0-107-generic'
user@xubuntu2004:~/linux-kernel-01$ ls -lt
total 448
-rw-rw-r-- 1 user user 216120 May 21 03:25 hack.ko
-rw-rw-r-- 1 user user 108928 May 21 03:25 hack.mod.o
-rw-rw-r-- 1 user user    0 May 21 03:25 Module.symvers
-rw-rw-r-- 1 user user  856 May 21 03:25 hack.mod.c
-rw-rw-r-- 1 user user   35 May 21 03:25 hack.mod
-rw-rw-r-- 1 user user   35 May 21 03:25 modules.order
-rw-rw-r-- 1 user user 108720 May 21 03:25 hack.o
-rw-rw-r-- 1 user user  154 May 19 20:42 Makefile
-rw-rw-r-- 1 user user   409 May 19 20:18 hack.c
user@xubuntu2004:~/linux-kernel-01$
```

As a result, after running the `make` command, you will find several new intermediate binaries. However, the most significant addition will be the presence of a new `hack.ko` file.

So, what's next. Run `dmesg` command in new terminal:

```
dmesg
```

```
Terminal - user@xubuntu2004: ~
File Edit View Terminal Tabs Help
[ 1739.961788] 14:51:28.304495 control Closing all guest files ...
[ 1739.964181] 14:51:28.306857 control vbgLR3GuestCtrlDetectPeekGetCancelSupport: Supported (#1)
[ 1740.877038] e1000: enp0s3 NIC Link is Down
[ 1740.877240] e1000 0000:00:03.0 enp0s3: Reset adapter
[ 1740.881893] usb 2-1: new full-speed USB device number 6 using ohci-pci
[ 1741.324752] usb 2-1: New USB device found, idVendor=80ee, idProduct=0021, bcdDevice= 1.00
[ 1741.324763] usb 2-1: New USB device strings: Mfr=1, Product=3, SerialNumber=0
[ 1741.324767] usb 2-1: Product: USB Tablet
[ 1741.324770] usb 2-1: Manufacturer: VirtualBox
[ 1741.337087] input: VirtualBox USB Tablet as /devices/pci0000:00/0000:00:06.0/usb2/2-1/2-1:1.0/0003:80EE:0021.0005/input/input11
[ 1741.393882] hid-generic 0003:80EE:0021.0005: input,hidraw0: USB HID v1.10 Mouse [VirtualBox USB Tablet] on usb-0000:00:06.0-1/input0
[ 1742.953670] e1000: enp0s3 NIC Link is Up 1000 Mbps Full Duplex, Flow Control: RX
[ 1743.217363] 21:23:34.720109 timesync vgsvcTimeSyncWorker: Radical host time change: 109 920 259 000 000ns (HostNow=1 718 918 614 720 000 000 ns HostLast=1 718 808 694 461 000 000 ns)
[ 1753.217959] 21:23:44.720694 timesync vgsvcTimeSyncWorker: Radical guest time change: 109 933 159 176 000ns (GuestNow=1 718 918 624 720 666 000 ns GuestLast=1 718 808 691 561 490 000 ns fSetTimeLastLoop=true )
user@xubuntu2004: ~$
```

Then run the following command from our `hack.ko` dir for load this module into running kernel:

```
sudo insmod hack.ko
```

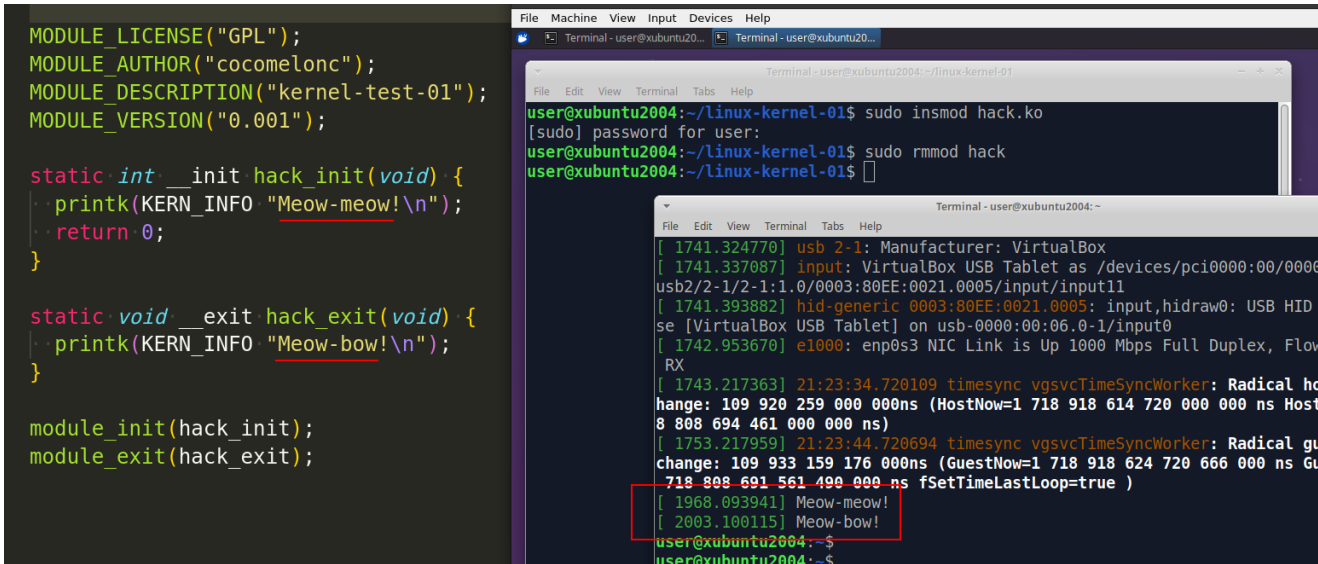
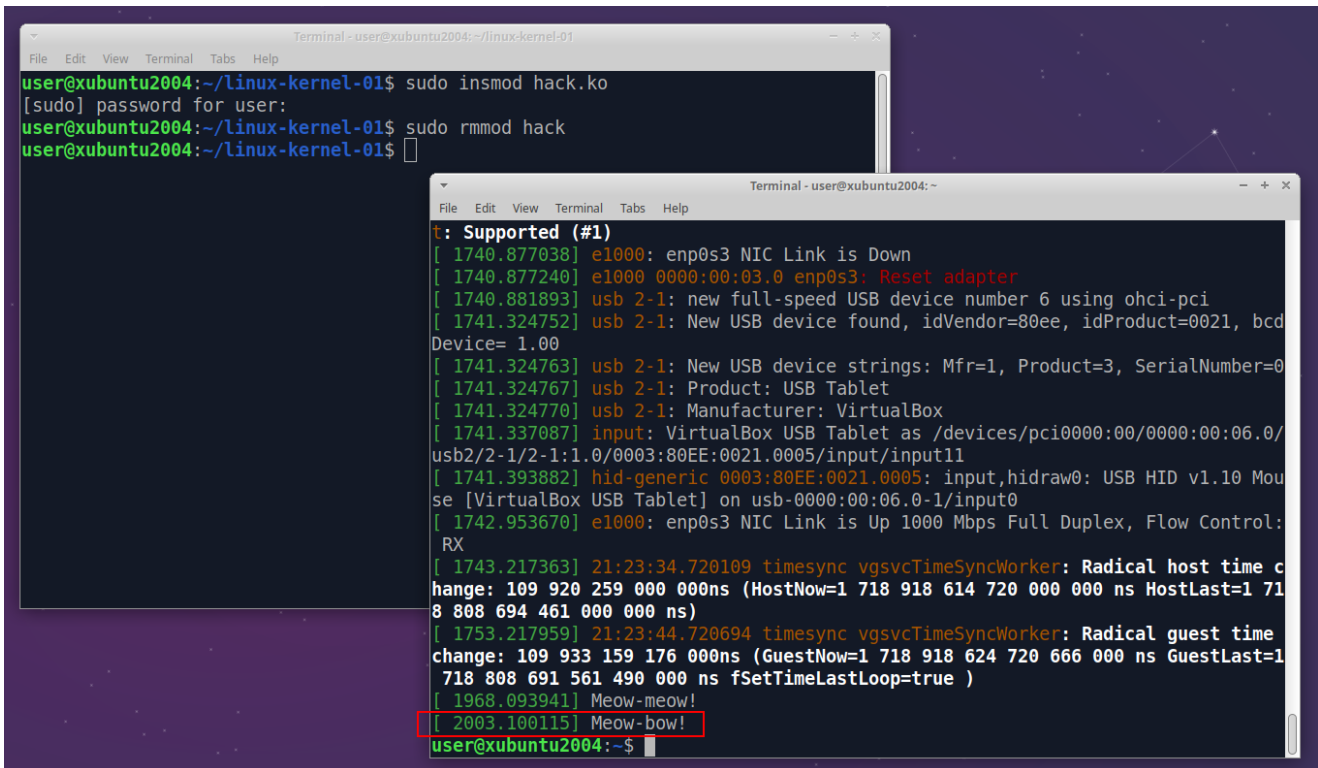
Now, if you check `dmesg` again from new terminal, you should see a `Meow-meow!` line:

```
Terminal - user@xubuntu2004: ~/linux-kernel-01
File Edit View Terminal Tabs Help
user@xubuntu2004:~/linux-kernel-01$ sudo insmod hack.ko
[sudo] password for user:
user@xubuntu2004:~/linux-kernel-01$

Terminal - user@xubuntu2004: ~
File Edit View Terminal Tabs Help
[ 1739.964181] 14:51:28.306857 control vbgLR3GuestCtrlDetectPeekGetCancelSupport: Supported (#1)
[ 1740.877038] e1000: enp0s3 NIC Link is Down
[ 1740.877240] e1000 0000:00:03.0 enp0s3: Reset adapter
[ 1740.881893] usb 2-1: new full-speed USB device number 6 using ohci-pci
[ 1741.324752] usb 2-1: New USB device found, idVendor=80ee, idProduct=0021, bcdDevice= 1.00
[ 1741.324763] usb 2-1: New USB device strings: Mfr=1, Product=3, SerialNumber=0
[ 1741.324767] usb 2-1: Product: USB Tablet
[ 1741.324770] usb 2-1: Manufacturer: VirtualBox
[ 1741.337087] input: VirtualBox USB Tablet as /devices/pci0000:00/0000:00:06.0/usb2/2-1/2-1:1.0/0003:80EE:0021.0005/input/input11
[ 1741.393882] hid-generic 0003:80EE:0021.0005: input,hidraw0: USB HID v1.10 Mouse [VirtualBox USB Tablet] on usb-0000:00:06.0-1/input0
[ 1742.953670] e1000: enp0s3 NIC Link is Up 1000 Mbps Full Duplex, Flow Control: RX
[ 1743.217363] 21:23:34.720109 timesync vgsvcTimeSyncWorker: Radical host time change: 109 920 259 000 000ns (HostNow=1 718 918 614 720 000 000 ns HostLast=1 718 808 694 461 000 000 ns)
[ 1753.217959] 21:23:44.720694 timesync vgsvcTimeSyncWorker: Radical guest time change: 109 933 159 176 000ns (GuestNow=1 718 918 624 720 666 000 ns GuestLast=1 718 808 691 561 490 000 ns fSetTimeLastLoop=true )
[ 1968.093941] Meow-meow!
user@xubuntu2004: ~$
```


For deleting our module from running kernel just run:

```
sudo rmmmod hack
```



As you can see, **Meow-bow!** message in kernel buffer, so everything is worked perfectly as expected! =^..^=

There are one more caveat of course. When building a Linux kernel module, it is important to note that it belongs to the specific kernel version it was built on. If you attempt to load a module onto a system with a different kernel, it is highly probable that it will fail to load.

I think we'll take a break here, we'll look at rootkits and stealers in the following posts.

I hope this post with practical example is useful for malware researchers, linux programmers and everyone who interested on linux kernel programming techniques.

[source code in github](#)

| This is a practical case for educational purposes only.

Thanks for your time happy hacking and good bye!

PS. All drawings and screenshots are mine