getting out the bin

We can download it like fricking losers from the link in or get if by force !!! `hexdump -Cv rootkit.ko` and then I run that script :

```
all = []
for line in x.split("\n"):
    line = line.strip()
    if line == "*":
        continue
    line = line.partition("|")[0]
    linex = line.partition(" ")[2]
    all += list(map(bytearray.fromhex , linex.split(" ")))

data = reduce(lambda x ,y : x+y , all)
print(*data,sep="\n")
with open("rootkit.ko", 'wb') as file :
    file.write(data)
```

where x is the stdout I copied from hexdump! and that is it . we have the bin :

```
00000000  7F 45 4C 46 01 01 01 00 00 00 00 00 00 00 00 00  .ELF............
00000010  01 00 03 00 01 00 00 00 00 00 00 00 00 00 00 00  ................
00000020  80 08 00 00 00 00 00 00 34 00 00 00 00 00 28 00  ........4.....(.
00000030  15 00 12 00 04 00 00 00 14 00 00 00 03 00 00 00  ................
00000040  47 4E 55 00 55 52 F8 93 CF 02 CE 13 E9 A1 83 AF  GNU.UR..........
00000050  3B 67 17 E8 84 49 3E AD 00 00 00 00 00 00 00 00  ;g...I>.........
00000060  55 89 E5 53 83 EC 0C E8 FC FF FF FF BA 00 00 00  U..S............
00000070  00 8B 5D 0C 89 D8 E8 FC FF FF FF 85 C0 75 1D 8B  ..].............u..
00000080  45 10 89 5C 24 04 89 44 24 08 8B 45 08 89 04 24  E..\$..D$..E...$
00000090  FF 15 00 00 00 00 83 C4 0C 5B 5D C3 C7 04 24 05  .........[]...$.
000000A0  00 00 00 E8 FC FF FF FF 83 C8 FF EB E9 8D 76 00  ..............v.
000000B0  55 89 E5 53 83 EC 08 E8 FC FF FF FF BA 00 00 00  U..S............
000000C0  00 8B 5D 08 89 D8 E8 FC FF FF FF 85 C0 75 16 8B  ..].............u..
000000D0  45 0C 89 1C 24 89 44 24 04 FF 15 00 00 00 00 83  E...$.D$........
000000E0  C4 08 5B 5D C3 C7 04 24 05 00 00 00 E8 FC FF FF  ..[]...$........
000000F0  FF 83 C8 FF EB E9 8D 76 00 8D BC 27 00 00 00 00  ........v...'....
00000100  55 89 E5 53 83 EC 10 E8 FC FF FF FF BA 00 00 00  U..S............
00000110  00 8B 5D 0C 89 D8 E8 FC FF FF FF 85 C0 75 24 8B  ..].............u$.
00000120  45 14 89 5C 24 04 89 44 24 0C 8B 45 10 89 44 24  E..\$..D$..E..D$
00000130  08 8B 45 08 89 04 24 FF 15 00 00 00 00 83 C4 10  ..E...$.........
00000140  5B 5D C3 C7 04 24 15 00 00 00 E8 FC FF FF FF 83  []...$..........
00000150  C8 FF EB E9 8D B6 00 00 00 00 8D BF 00 00 00 00  ................
```

let watch the ida code

```
mov     eax, ds:0C15FA034h
```

4	write	man/ cs/	4	unsigned int fd	const char *buf	size_t count	-	-	-

```
mov     ds:sys_open, eax
```

4	write	man/ cs/	4	unsigned int fd	const char *buf	size_t count	-	-	-

and as we can see -> hex(5 * 4(size of entry)) = 20 so that is pretty neat

and we can see that it is done to the rest :

```
mov     eax, ds:0C15FA4BCh
mov     ds:sys_openat, eax
mov     eax, ds:0C15FA16Ch
mov     ds:sys_symlink, eax
mov     eax, ds:0C15FA4E0h
mov     ds:sys_symlinkat, eax
mov     eax, ds:0C15FA044h
mov     ds:sys_link, eax
mov     eax, ds:0C15FA4DCh
mov     ds:sys_linkat, eax
mov     eax, ds:0C15FA0B8h
mov     ds:sys_rename, eax
mov     eax, ds:0C15FA4D8h
mov     ds:sys_renameat, eax
```

- syscall table base: 0C15FA020
- 0C15FA4BCh = base + 0x49c = entry 295

295	openat	man/ cs/	127	int dfd	const char *filename	int flags	umode_t mode	-	-

- 0C15FA16Ch = base + 0x14c = entry 83

83	symlink	man/ cs/	53	const char *old	const char *new	-	-	-	-

- 0C15FA4E0h = base + 0x4c0 = entry 304

304	symlinkat	man/ cs/	130	const char * oldname	int newdfd	const char * newname	-	-	-

- 0C15FA044h = base + 0x24 = entry 9

9	link	man/ cs/	9	const char *oldname	const char *newname	-	-	-	-

- 0C15FA4DCh = base + 0x4bc = entry 303

305	readlinkat	man/ cs/	131	int dfd	const char *path	char *buf	int bufsiz	-	-

- 0C15FA0B8h = base + 0x98 = entry 38

38	rename	man/ cs/	26	const char *oldname	const char *newname	-	-	-	-

- 0C15FA4D8h = base + 0x4b8 = entry 302

302	renameat	man/ cs/	12E	int olddfd	const char * oldname	int newdfd	const char * newname	-	-

so as we can see I told you some stuff but what about the kaslr ? and how does I know that those address are really true ???

so lets answer you :

- let use cmdline to check if it is on

```
/ # cat /proc/cmdline
root=/dev/ram rw console=ttyS0 rdinit=/bin/ash
```

and lets check one from machine with kaslr :

```
idang@idang ~> cat /proc/cmdline
BOOT_IMAGE=/vmlinuz-5.15.0-78-generic root=/dev/mapper/ubuntu--vg-ubuntu--lv ro
```

Kernel Address Space Layout Randomisation

Kernel Address Space Layout Randomisation (kASLR) aims to make some kernel exploits more difficult to implement by randomizing the base address value of the kernel. Exploits that rely on the locations of internal kernel symbols must discover the randomized base address.

kASLR is available starting with Ubuntu 14.10 and is enabled by default in 16.10 and later.

Before 16.10, you can specify the "kaslr" option on the kernel command line to use kASLR.

Note: Before 16.10, enabling kASLR will disable the ability to enter hibernation mode.

for the second question : I just printed : `cat /proc/kallsyms` and searched for intresing stuff - not a big deal at all!!!

so let continue to dissamble :

```
mov     dword ptr [eax+14h], offset sys_open_hooked
mov     dword ptr [eax+49Ch], offset sys_openat_hooked
mov     dword ptr [eax+14Ch], offset sys_symlink_hooked
mov     dword ptr [eax+4C0h], offset sys_symlinkat_hooked
mov     dword ptr [eax+24h], offset sys_link_hooked
mov     dword ptr [eax+4BCh], offset sys_linkat_hooked
mov     dword ptr [eax+98h], offset sys_rename_hooked
mov     dword ptr [eax+4B8h], offset sys_renameat_hooked
```

as you can see we hook all the functions .

> note : before and after we call the wp function to set the write protection to the page of she syscall table on and off

let's analyze in what are the commands :

```
.text:08000250 sys_open_hooked proc near               ; DATA XREF: initmodule+69↓o
```

and the opcodes of the mov on the syscall table are :

```
C7 40 14 50 02 00 08          mov     dword ptr [eax+14h], offset sys_open_hooked
```

> note : remmber that the kaslr is not working !!!

so if we will return the open command to the original one that located in :

```
/proc # cat kallsyms | grep sys_open
c106c7c0 W compat_sys_open_by_handle_at
c1158bc0 T do_sys_open
c1158d70 T sys_open
c1158db0 T sys_openat
c11a37b0 T sys_open_by_handle_at
c11b47d0 t proc_sys_open
```

`c1158d70`

and formatted :

`70 8D 15 C1`

it will return to work normaly

The plan :

create a new LKM that based on rootkit.ko that will change the open to the original one and then we will be able in theory to run the cat command .

so let's validate in Ida that the byte sequence of the little endian of the address do not show up any where else `50 02 00 08`

so let's run a few commands (I recommend to open new ssh session)

so let's patch with ida what we wanted and see the diff

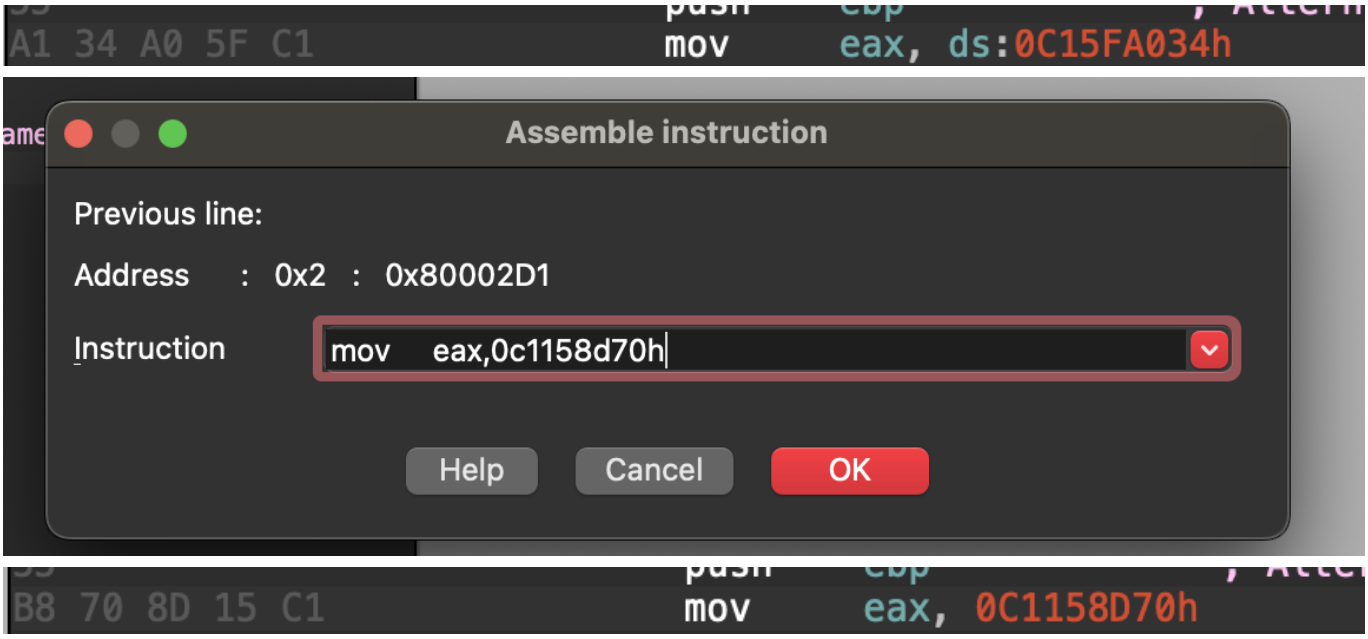
```
idang@Idans-MacBook-Air rootkit % hexdump -C footkit.ko > idk1.hex
idang@Idans-MacBook-Air rootkit % hexdump -C footkit.ko.bak > idk2.hex
```

```
idang@Idans-MacBook-Air rootkit % diff idk1.hex idk2.hex
58c58
< 00000390  fc ff ff ff a1 00 00 00  00 c7 40 14 70 8d 15 c1  |..........@.p...|
---
> 00000390  fc ff ff ff a1 00 00 00  00 c7 40 14 00 00 00 00  |..........@.....|
```

`A1 34 A0 5F C1`

in

`B8 70 8D 15 C1`



Commands

```
cp rootkit.ko footkit.ko

sed -i $'s/\xA1\x34\xA0\x5F\xC1/\xB8\x70\x8D\x15\xC1/g' footkit.ko

sed -i 's/root/foot/g' footkit.ko

sed -i 's/flag/idan/g' footkit.ko

insmod footkit.ko
```

and then decompress the file with gzip :

```
import base64
import gzip

flag="""H4sIAMdtslUAA+3PMQrCQBCF4alzim3tZnY3K3gFO3OCFIlIAoG4It7eNaVFrEIQ/q
95DDPFm35s
r7IxLVKMSxbfqRq8mA91sGMyn0RNo5k43brYx+Oe29k5macpr9392v+pS3lruOX8cs2z60qcl+
F0
qPZuBgAAAAAAAAAAAAAAABY8wZDzE0OACgAAA=="""

print(gzip.decompress(base64.b64decode(flag)))
```

Important Reosurces :

- syscall table
- ubuntu sec fetures